



فصلنامه علمی پژوهش‌های سایبرالکترونیک

صاحب امتیاز: دانشکده علوم و مهندسی دفاعی دانشگاه افسری و تربیت پاسداری امام حسین (ع)

سر دبیر: دکتر محمدعلی جوادزاده

مدیر مسئول: دکتر محمد کاظم بصیرتی

اعضای هیئت تحریریه

دکتر محمد کاظم بصیرتی	استادیار دانشگاه
دکتر حمیدرضا لشکریان	استادیار دانشگاه
دکتر مسعود باقری	دانشیار دانشگاه
دکتر محمدعلی جوادزاده	استادیار دانشگاه
دکتر حمید باباخانیان	مدرس دانشگاه
دکتر ضرغام رستمی	استادیار دانشگاه
دکتر حسین صابری	استادیار دانشگاه
دکتر مهدی علی‌نژاد	استادیار دانشگاه
مهندس سید محمود قطبانی	مدرس دانشگاه
مهندس علیرضا صفایی	مدرس و عضو هیئت علمی دانشگاه
مهندس حامد شیروی	مدرس دانشگاه
مهندس مرتضی یزدان‌پناه	مدرس دانشگاه
مهندس یوسف برجعلی	مدرس دانشگاه



همکاران علمی و داوران این شماره

دکتر منصور احدنژاد، دکتر حمید اکبری، دکتر رضا جلایی، دکتر بهروز خادم، دکتر علی کریمی، دکتر مصطفی عباسی،
دکتر مهدی علی‌نژاد، دکتر مهدی نقوی، مهندس علی بهتری، مهندس حامد شیروی، مهندس رامین دلیر،
مهندس سامان کشوری، مهندس پیمان قنوازی، مهندس مجید حاج جواد

مدیر اجرایی: مهندس علیرضا صفایی

ویراستار ادبی: سرکار خانم مهندس طوفانی

ویراستار انگلیسی و مترجم: مهندس فرامرز شمس سوسه‌باب

حروفچین و صفحه‌آرا: مهندس مرتضی یزدان‌پناه

طراح جلد: مهندس علی ذوالفقاریان

شمارگان: ۱۰۰ (نسخه چاپی) ۲۰۰ (نسخه الکترونیکی) قیمت نسخه چاپی:

نشانی: تهران، بزرگراه شهید بابایی، بعد از پل لشکرک، دانشگاه افسری و تربیت پاسداری امام حسین (ع)، دانشکده علوم و مهندسی دفاعی، تلفن: ۷۷۱۰۳۰۵۰ داخلی ۱۰۹

Web Site: www.Jce.ihuo.ac.ir

E-mail: Jce@ihuo.ac.ir

تمام حقوق مادی و معنوی برای دانشکده علوم و مهندسی دفاعی محفوظ است.

فهرست

- ❖ سخن سردیر..... (یازده)
- ❖ تقسیم‌بندی معنایی برای تصاویر پهپاد با استفاده از روش‌های یادگیری عمیق..... ۱۳
 - سید محمدعلی ضیائی
- ❖ کاربرد سطوح انتخاب‌گر فرکانسی پهن باند در الکترونیک دفاعی..... ۲۷
 - علی عزیزی
- ❖ بررسی ارتباطات بی‌سیم برای شبکه هوشمند مبتنی بر رویکردها و فناوری‌های نوپدید از دیدگاه EMC..... ۳۷
 - ضرغام رستمی، احمد عطائی
- ❖ تحلیل و شبیه‌سازی ساختار نیم جمع‌کننده‌های نوری مبتنی بر بلورهای فوتونی و ارائه پیشنهادهایی جهت بهبود عملکرد در مدارهای مجتمع نوری..... ۴۷
 - محمد مقصودی
- ❖ مزایای استفاده از جنگ سایر الکترونیک در محیط‌های عملیاتی..... ۵۷
 - مهرداد نجفی نوکاشتی، محمدرضا موحدی‌صفت
- ❖ ارائه یک دسته‌بندی بر اساس مشخصه‌های سیستم عامل‌های بی‌درنگ..... ۷۱
 - مهدی قزلسفلو، مهدی نقوی

ارائه یک دسته‌بندی بر اساس مشخصه‌های سیستم‌عامل‌های بی‌درنگ

مهدی قزلسفلو^۱، مهدی نقوی^۲

چکیده

سیستم‌عامل بی‌درنگ سیستم‌عاملی است که زمان در آن، عامل کلیدی است. زمانی از یک سیستم بی‌درنگ استفاده می‌شود که لازم باشد عملیات پردازش یا جریان داده در زمان کاملاً مشخصی انجام شود. یک سیستم بی‌درنگ اغلب به‌عنوان ابزار کنترلی در یک کاربرد خاص استفاده می‌شود. ساختار پایه سیستم‌عامل بی‌درنگ شبیه سیستم‌عامل‌های معمولی است که علاوه بر خصوصیات آن، امکان مدیریت زمانی کارها را با کمک متدولوژی‌هایی مانند هماهنگ‌سازی، مدیریت وقفه، زمان‌بندی، مدیریت حافظه و... فراهم می‌کند که منجر به استفاده گسترده سیستم‌عامل بی‌درنگ در سیستم‌های کنترل پرواز، سامانه‌های هدایت موشکی، سیستم پزشکی، سیستم‌های کنترل راکتور هسته‌ای، سیستم خلبان خودکار هواپیما، سیستم ضد موشکی و سایر سیستم‌های کنترلی حساس می‌شوند. این نوع سیستم‌عامل کاربرد قابل توجهی در صنعت دارد؛ لذا تدوین و ارائه یک طبقه‌بندی مطابق با آخرین بررسی‌ها و دستاوردهای این حوزه کمک شایانی به اندیشمندان این حوزه خواهد کرد. در این مقاله ضمن تدوین و ارائه این طبقه‌بندی، به بررسی جامع سیستم‌عامل بی‌درنگ و موضوعات آن شامل هماهنگ‌سازی، مدیریت وقفه، سطوح اولویت بی‌درنگ، مدیریت حافظه، زمان‌بندی، سمافور، پیش‌پرداخت سریع کار، امنیت و پرکاربردترین سیستم‌عامل‌های بی‌درنگ است.

کلید واژه‌ها: سیستم‌عامل بی‌درنگ، زمان‌بندی، مدیریت حافظه، هماهنگ‌سازی، مدیریت وقفه، سمافور

۱- دانشجوی کارشناسی ارشد مهندسی نرم‌افزار

۲- استادیار دانشکده کامپیوتر دانشگاه جامع امام حسین (ع) mnaghavi@ihu.ac

۱- مقدمه

سیستم بی درنگ به سیستمی گفته می شود که درستی منطقی خروجی ها و هم به موقع دریافت کردن آنها تضمین کند [1]. سیستم های بی درنگ به دودسته سیستم های سخت و نرم طبقه بندی می شوند. یک سیستم بی درنگ سخت یک سیستم کاملاً قطعی و با محدودیت زمانی است. مانند سیستم های کنترل پرواز، سامانه های هدایت موشکی، سیستم پزشکی، سیستم های کنترل راکتور هسته ای، سیستم خلبان خودکار در هواپیما، سیستم ضد موشکی. به عبارت دیگر اگر در یک سیستم، نقض یک ضربالاجل زمانی با احتمال آسیب یا فاجعه بزرگ همراه باشد، آن را به عنوان یک سیستم بی درنگ سخت [2] می نامند.

سیستم بی درنگ نرم سیستمی است که در صورت عدم تولید نتایج مطابق با نیاز زمان بندی مشخص، عملکرد آن کاهش می یابد. در یک سیستم بی درنگ نرم، جلسه مهلت برای هر کار اجباری نیست، اما فرایند باید پردازش شود و نتیجه را ارائه دهد؛ مانند کامپیوتر شخصی، سیستم های صوتی و تصویری، سیستم های پایش آب و هوا، وب گردی.

مشاهده می کنیم که انتخاب یک سیستم عامل در طراحی یک سیستم بی درنگ مهم است. طراحی یک سیستم بی درنگ شامل انتخاب زبان مناسب، تقسیم بندی و ادغام وظایف و تعیین اولویت ها برای مدیریت زمان پاسخ است. بسته به اهداف زمان بندی، موازی سازی و ارتباطات ممکن است متعادل باشند. طراح باید وظایف حیاتی را تعیین کند و اولویت های بالایی را به آنها اختصاص دهد. با این حال، باید مراقب بود که از گرسنگی جلوگیری شود، گرسنگی زمانی رخ می دهد که وظایف با اولویت بالاتر همیشه آماده اجرا هستند و در نتیجه زمان پردازشگر کافی برای وظایف با اولویت پایین تر وجود ندارد. اگر کاری وجود دارد که نمی توان بدون ایجاد خرابی سیستم، از وقفه های بدون اولویت جلوگیری کرد. در حالت ایده آل، کنترل کننده وقفه باید زمینه را ذخیره کند، وظیفه ای ایجاد کند که وقفه را سرویس کند و کنترل را به سیستم عامل بازگرداند.

هنگامی که یک سیستم عامل بی درنگ انتخاب می شود، باید چندین معیار مانند هزینه های مجوز، رد پای حافظه، قابلیت پیش بینی و تأخیر در رسیدگی به عملیات حیاتی ناشی از رویدادهای داخلی یا خارجی را در نظر بگیرد. این تأخیر همچنین می تواند تحت تأثیر زمان مورد نیاز برای عملیات تعویض زمینه کار قرار گیرد [3].

استفاده از یک وظیفه برای انجام بخش عمده ای از سرویس وقفه، امکان انجام سرویس را بر اساس اولویت انتخاب شده توسط طراح فراهم می کند و به حفظ سیستم اولویت در سیستم عامل بی درنگ کمک می کند. علاوه بر این، زمان های پاسخ خوب ممکن است به رد پای حافظه کوچک در سیستم های ضعیف تر از منابع نیاز داشته باشد. واضح است که انتخاب یک سیستم عامل بی درنگ در فرایند طراحی برای پشتیبانی از اولویت ها، وقفه ها، تایمرها، ارتباطات بین وظیفه، همگام سازی، پردازش چندگانه و مدیریت حافظه مهم است. ساختار این مقاله به شرح زیر است. بخش ۲ الزامات اساسی یک سیستم عامل بی درنگ را تشریح می کند. بخش ۳ ویژگی های سیستم عامل بی درنگ را بررسی می کند. بخش ۴ تفاوت سیستم عامل بی درنگ با سیستم عامل معمولی را مورد بررسی قرار می دهد. بخش ۵ سیستم عامل های محبوب در سال ۲۰۲۱ را معرفی می کند.

۲- طبقه‌بندی سیستم‌های عامل بی‌درنگ

با بررسی و مطالعه مقالات مختلف در زمینه طبقه‌بندی سیستم‌های عامل بی‌درنگ، یک طبقه‌بندی جدید جامع گردآوری و تدوین شده است که در شکل (۱) نشان داده می‌شود. بر اساس این طبقه‌بندی نیازمندی‌ها، مشخصه‌ها و فنون به کاررفته سیستم‌عامل‌های بی‌درنگ تشریح خواهند شد.

۳- نیازمندی‌های سیستم‌عامل بی‌درنگ

نیازمندی‌ها و مشخصه‌های اصلی سیستم‌عامل بی‌درنگ شامل ۵ مورد زیر است که هر یک را به اختصار توضیح می‌دهیم.

۱-۱- چند وظیفه‌ای و قابل پیشگیری

چند وظیفه‌ای و قابل پیشگیری به این معنی است که برای پشتیبانی از چندین کار در برنامه‌های بی‌درنگ، یک سیستم‌عامل بی‌درنگ باید چند وظیفه‌ای و قابل جایگزین کردن وجود داشته باشد. برنامه‌ریز باید بتواند هر وظیفه‌ای را از سیستم گرفته و منع را به کاری که بیشتر به آن نیاز دارد بدهد. یک سیستم‌عامل بی‌درنگ همچنین باید چندین سطح وقفه را مدیریت کند تا چندین سطح اولویت را مدیریت کند.

۱-۲- تشخیص پویای مهلت زمانی

تشخیص پویای مهلت زمانی به این معنی است که به منظور دستیابی به اولویت نسبت به دیگران، یک سیستم‌عامل بی‌درنگ باید بتواند به صورت پویا کار را با اولین مهلت شناسایی کند. برای رسیدگی به مهلت‌ها، اطلاعات مهلت ممکن است به سطوح اولویتی تبدیل شوند که برای تخصیص منابع استفاده می‌شوند. اگرچه چنین رویکردی مستعد خطا است، با این وجود به دلیل عدم وجود راه حل بهتر مورد استفاده قرار می‌گیرد.

۱-۳- همگام‌سازی قابل پیش‌بینی

برای اینکه چندین رشته^۴ با یکدیگر به موقع ارتباط برقرار کنند، مکانیسم‌های ارتباط بین وظیفه‌ای و هماهنگ‌سازی قابل پیش‌بینی^۵ مورد نیاز است [4]. یکپارچگی معنایی و همچنین به موقع بودن، قابل پیش‌بینی است. همگام‌سازی قابل پیش‌بینی نیاز به مصالحه دارد. قابلیت بستن/باز کردن منابع یکی از راه‌های دستیابی به یکپارچگی داده‌ها است.

۱-۴- سطوح اولویت کافی

^۱Taxonomy

^۲Multi-tasking and preemptable

^۳Dynamic deadline identification

^۴Thread

^۵Predictable synchronization

هنگام استفاده از زمان‌بندی وظایف اولویت‌دار، سیستم‌عامل بی‌درنگ باید دارای تعداد سطوح اولویت کافی^۱ برای اجرای مؤثر داشته باشد [5]. وارونگی اولویت، زمانی اتفاق می‌افتد که یک کار با اولویت بالاتر باید منتظر یک کار با اولویت پایین‌تر برای انتشار یک منبع باشد. دو راه حل در برخورد با وارونگی اولویت، یعنی وراثت اولویت و پروتکل‌های سقف اولویت^۲، به سطوح اولویت کافی نیاز دارند در مکانیزم ارث‌بری اولویت، وظیفه‌ای که یک کار با اولویت بالاتر را مسدود می‌کند، اولویت بالاتر را برای مدت‌زمان به ارث می‌برد. زمان‌بندی اولویت کار دسترسی را برابر با اولویت منبع قرار می‌دهد. پس از اینکه یک کار یک منبع را آزاد کرد، اولویت آن به مقدار اولیه آن برمی‌گردد. با این حال، وقتی اولویت کاری برای دسترسی به یک منبع افزایش می‌یابد، نباید منتظر منبع دیگری می‌بود.

۵-۱- تأخیرهای از پیش تعریف شده^۳

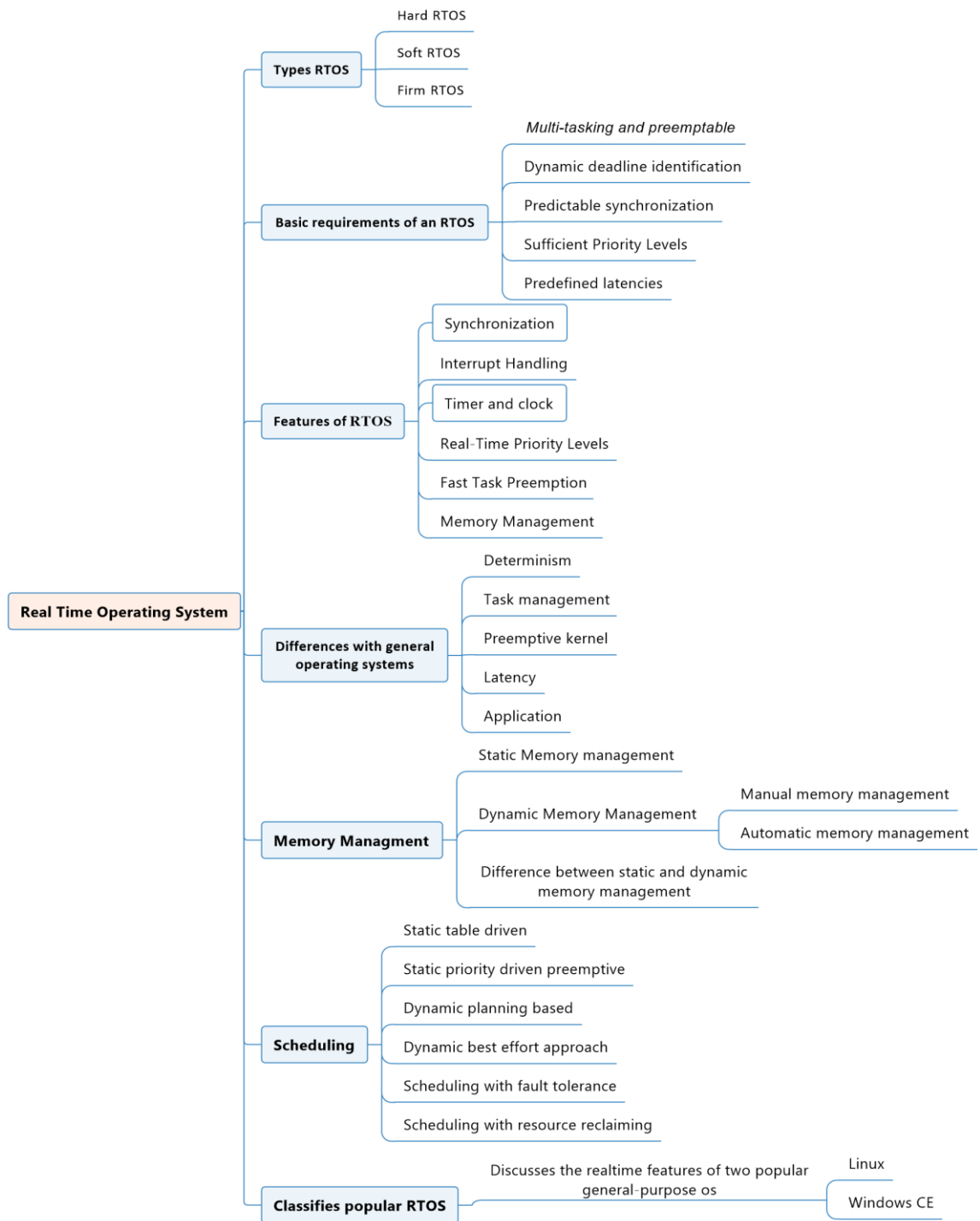
تأخیرهای از پیش تعریف شده شامل موارد زیر هستند:

تأخیر تغییر وظیفه: زمان موردنیاز برای ذخیره زمینه یک کار در حال اجرای فعلی و جابه‌جایی به کار دیگر.
تأخیر وقفه: زمان سپری شده بین اجرای آخرین دستور کار قطع شده و اولین دستورالعمل در کنترل‌کننده وقفه [6].

^۱Sufficient Priority Levels

^۲ PCP

^۳Predefined latencies



شکل ۱- طبقه‌بندی نیازمندی‌ها، مشخصه‌ها و فنون به کار رفته در سیستم‌عامل‌های بی‌درنگ

تأخیر ارسال وقفه: زمان از آخرین دستورالعمل در کنترل کننده وقفه تا کار بعدی که برای اجرا برنامه ریزی شده است. را برابر با اولویت منبع قرار می دهد. پس از اینکه یک کار یک منبع را آزاد کرد، اولویت آن به مقدار اولیه آن برمی گردد. با این حال، وقتی اولویت کاری برای دسترسی به یک منبع افزایش می یابد، نباید منتظر منبع دیگری می بود.

۴- ویژگی ها

نیازمندی ها و پیش نیازهای سیستم عامل بی درنگ شامل موارد زیر است که هر یک را به اختصار توضیح داده می شود.

۴-۱- همگام سازی

همگام سازی^۱ برای کارهای بی درنگ برای به اشتراک گذاشتن منابع منحصربه فرد متقابل ضروری است. برای اینکه چندین رشته با یکدیگر به موقع ارتباط برقرار کنند، مکانیسم های ارتباط بین وظیفه ای و هماهنگ سازی قابل پیش بینی مورد نیاز است [7].

۴-۲- مدیریت وقفه

در مدیریت وقفه^۲ روال سرویس وقفه برای رسیدگی به وقفه استفاده می شود. تأخیر وقفه به عنوان تأخیر زمانی بین وقوع وقفه و اجرای روال سرویس وقفه مربوطه تعریف می شود.

۴-۳- تایمر و ساعت

خدمات ساعت و تایمر^۳ با وضوح کافی بخش مهمی از هر سیستم عامل بی درنگ است. یک سیستم بی درنگ توزیع شده شامل مجموعه ای از گره ها است که فقط از طریق ارسال پیام با یکدیگر ارتباط برقرار می کنند.

گره ها کامپیوترهایی هستند که عملکرد مشخصی را انجام می دهند. هر گره حاوی یک ساعت بلادرنگ محلی بادقت کوارتز است. برای اندازه گیری زمان وقوع رویدادها و مدت زمان فواصل بین رویدادها که از گره های مختلف مشاهده می شود، لازم است یک مرجع زمانی مشترک بادقت مشخص به برنامه ارائه شود. این مرجع زمانی رایج که به عنوان «زمان جهانی تقریبی» شناخته می شود، از زمان های محلی ساعت های بی درنگ محلی با استفاده از یک الگوریتم همگام سازی ساعت ایجاد می شود.

الگوریتم های همگام سازی ساعت برای سیستم های بی درنگ توزیع شده باید الزامات زیر را برآورده کنند.

^۱Synchronization

^۲Interrupt Handling

^۳ISR

^۴Timer and clock

- (۱) حداکثر اختلاف مقادیر زمانی هر رویداد منفرد مشاهده شده از دو گره مختلف که در متریک پایگاه زمانی محلی این دو گره از سیستم توزیع شده اندازه‌گیری می‌شود، باید توسط یک ثابت شناخته شده محدود شود.
- (۲) زمان تقریبی جهانی باید کرونوسکوپی باشد، یعنی می‌توان از آن برای اندازه‌گیری دقیق فواصل زمانی کوچک در هر نقطه از زمان استفاده کرد.
- (۳) الگوریتم همگام‌سازی ساعت باید قابل تحمل خطا باشد، به‌عنوان مثال، یک خطا در هر ساعت محلی یا از دست دادن یک پیام باید تحمل شود.
- (۴) اجرای الگوریتم همگام‌سازی ساعت نباید عملکرد کل سیستم را کاهش دهد [8].

۴-۴- سطوح اولویت بی‌درنگ

یک سیستم‌عامل بی‌درنگ باید از سطوح اولویت بی‌درنگ پشتیبانی کند تا زمانی که برنامه‌نویس یک مقدار اولویت را به یک کار اختصاص می‌دهد، سیستم‌عامل به‌خودی‌خود آن را تغییر ندهد.

۴-۵- ممانعت به دلیل وظایف اولویت‌دار

ممانعت به دلیل وظایف اولویت‌دار به این معنی است که در سیستم‌عامل‌های بی‌درنگ وقتی یک وظیفه با اولویت بالاتر نیاز به اجرا داشته باشد، وظیفه با اولویت پایین متوقف شده و وظیفه با اولویت بالاتر به سرعت اجرا می‌شود. این امر موجب می‌شود که سیستم به رویدادهای مهم، سریع پاسخ دهد [7].

۴-۶- مدیریت حافظه

یک سیستم‌عامل بی‌درنگ برای اینکه بتواند پاسخگویی سریع‌تری داشته باشد، از حافظه کوچکی، آن هم فقط برای عملکردهای ضروری یک برنامه استفاده می‌کند [9] [10]. به‌طورکلی، مدیریت حافظه سیستم‌عامل بی‌درنگ را می‌توان به‌عنوان مدیریت حافظه ایستا^۴ و مدیریت حافظه پویا^۵ طبقه‌بندی کرد [11]. در ادامه به توضیح هر کدام می‌پردازیم.

مدیریت حافظه ایستا وظایفی را با فضای داده موقت فراهم می‌کند. حافظه آزاد سیستم به مجموعه‌ای از بلوک‌های حافظه با اندازه ثابت تقسیم می‌شود که می‌توان آن‌ها را توسط وظایف درخواست کرد. هنگامی که یک کار با استفاده از یک بلوک حافظه تمام می‌شود، باید آن را به فضای داده برگرداند. راه دیگر برای فراهم کردن فضای موقت برای کارها از طریق اولویت‌ها است. یک فضای داده حافظه به وظایف با اولویت بالا و دیگری به وظایف با

^۱Real-Time Priority Levels

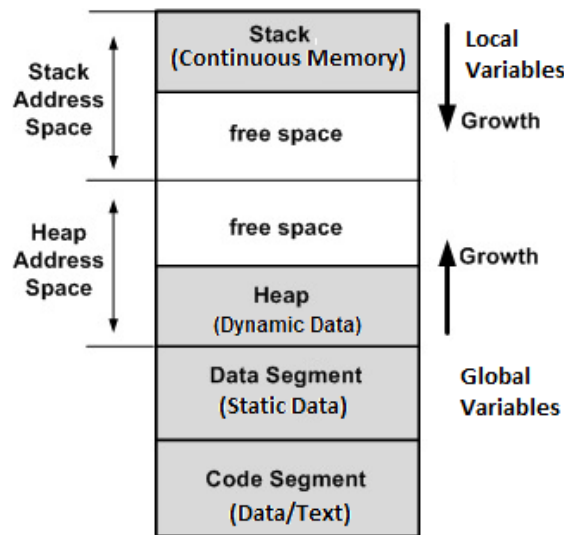
^۲Fast Task Preemption

^۳Memory Management

^۴Static memory management

^۵Dynamic memory management

اولویت پایین اختصاص داده شده است. اندازه فضای داده با اولویت بالا به گونه‌ای است که در بدترین حالت نیاز به حافظه سیستم داشته باشد. به فضای داده با اولویت پایین، حافظه آزاد باقی‌مانده داده می‌شود. اگر وظایف با اولویت پایین، فضای داده حافظه با اولویت پایین را خسته کنند، باید منتظر بمانند تا حافظه قبل از اجرای بیشتر به فضای داده بازگردانده شود. شکل ۲ چیدمان قطعات حافظه که سیستم عامل در اختیار برنامه قرار می‌دهد را نشان می‌دهد.



شکل ۲- چیدمان حافظه در اختیار یک برنامه

اندازه قطعه داده متغیرهای سراسری ثابت است و از نوع حافظه ایستا قلمداد می‌شود.

قطعه حافظه پشته برای متغیرهای محلی زیربرنامه‌ها و توابع در نظر گرفته شده است. در این حافظه اطلاعات پشت‌سرهم و به ترتیب قرار می‌گیرند به این صورت که آخرین داده ذخیره شده در بالای پشته قرار می‌گیرد و به اصطلاح **push** می‌شود، حال اگر قصد برداشتن اطلاعات یا به اصطلاح **pop** کردن اطلاعات را داشته باشیم آخرین اطلاعات وارد شده در پشته را در اختیار داریم از آنجاکه در حافظه پشته نیازی به پیدا کردن فضای خالی در حافظه نیست و محل قرارگیری اطلاعات مشخص است (بالای حافظه) بنابراین حافظه سریع تر از حافظه **heap** است.

حافظه **Heap** مربوط به زمان اجرا است و فضای اشغال شده در **heap** با اتمام کار تابع آزاد نمی‌شوند و تا زمانی که زباله‌روب این فضا را آزاد کند یا توسط برنامه‌نویس داده‌ها از حافظه پاک نشوند در این فضا باقی می‌ماند. اندازه

^۱Data Segment

^۲Global Variables

^۳Runtime

حافظه heap متغیر است به همین دلیل به آن حافظه پویا گفته می‌شود.

در این نوع از حافظه برای ذخیره مقادیر ابتدا محاسبه‌ای توسط سیستم‌عامل صورت می‌گیرد تا اولین فضای حافظه‌ای که اندازه آن متناسب با اندازه‌ای که موردنیاز ماست را پیدا کند، در صورت وجود این میزان از حافظه درخواستی آن را به صورت رزرو شده درمی‌آورد تا بقیه برنامه‌ها به این فضا دسترسی نداشته باشند، سپس آدرس ابتدای این فضای محاسبه شده به صورت یک اشاره‌گر آدر اختیارمان قرار می‌دهد.

متغیرها به صورت پیش‌فرض در این حافظه قرار نمی‌گیرند و اگر قصد ذخیره متغیرها در این حافظه را داشته باشیم باید به صورت دستی این اقدام انجام شود. متغیرهایی که در heap ذخیره می‌شوند به طور خودکار حذف نمی‌شوند و باید توسط برنامه‌نویس و به صورت فراخوانی حذف شوند. به‌طورکلی مدیریت حافظه heap توسط برنامه‌نویس انجام می‌شود. آرایه‌های پویا در heap ذخیره می‌شوند.

جدول ۱- مدیریت حافظه پویا و ایستا

مدیریت حافظه ایستا	مدیریت حافظه پویا
تخصیص حافظه در زمان کامپایل یا طراحی انجام می‌شود.	تخصیص حافظه در زمان اجرا یا در حین اجرا انجام می‌شود.
تخصیص حافظه ایستا یک فرایند ثابت است که به این معنی است که حافظه موردنیاز برای یک فرایند خاص از قبل شناسایی شده است و پس از تخصیص حافظه، هیچ تغییری در طول اجرا نمی‌توان انجام داد.	تخصیص حافظه دینامیک به مدیر حافظه نیاز دارد تا مشخص کند کدام قسمت از حافظه تخصیص‌یافته و کدام یک تخصیص داده نشده است؛ بنابراین، هنگامی که یک فرایند درخواست حافظه می‌کند، می‌تواند هر زمان که نیاز باشد حافظه را تخصیص داده و به آن اختصاص دهد.
تخصیص و عدم تخصیص حافظه در حین اجرا انجام نمی‌شود.	اتصالات حافظه در حین اجرا ایجاد و از بین می‌روند.
فضای حافظه بیشتر از فضای موردنیاز است.	فضای حافظه تقریباً اندازه فضای موردنیاز است.

مدیریت حافظه پویا به دودسته خودکار و دستی تقسیم‌بندی می‌شود:

مدیریت دستی حافظه^۳: در مدیریت دستی حافظه، توسعه‌دهنده کنترل مستقیم بر تخصیص حافظه دارد. به طور معمول، این یا توسط توابع مدیریت پشته یا توسط ساختارهای زبانی که روی پشته تأثیر می‌گذارند انجام می‌شود. مزیت مدیریت حافظه دستی این است که درک و استفاده از آن برای توسعه‌دهندگان ساده است. با این حال،

^۱Dynamic memory

^۲pointer

^۳Manual memory management

نقطه ضعف این است که خطاهای مدیریت حافظه بسیار معمولی هستند.

در مدیریت دستی حافظه، برنامه‌نویس کنترل مستقیمی بر فرایند تخصیص و بازیافت حافظه دارد. معمولاً این با فراخوانی صریح به توابع مدیریت پشته یا توسط ساختارهای زبانی است که روی پشته تأثیر می‌گذارد.

مزیت: مدیریت دستی حافظه برای برنامه‌نویسان برای درک و استفاده آسان‌تر است.

عیب: اشکالات مدیریت حافظه زمانی که از مدیریت دستی حافظه استفاده می‌شود رایج است.

مدیریت خودکار حافظه^۱: در مدیریت خودکار حافظه، مدیر حافظه خودکار معمولاً کار خود را با استفاده مجدد (بازیافت) از بلوک‌های حافظه است که خارج از محدوده اشیاء برنامه هستند (معمولاً کار خود را با بازیافت بلوک‌هایی که از متغیرهای برنامه غیرقابل دسترسی هستند، انجام می‌دهند). به این تخصیص‌دهنده‌های خودکار حافظه، جمع‌کننده زباله نیز گفته می‌شود. مزیت استفاده از آن این است که اکثر خطاهای مدیریت حافظه را حذف می‌کند. اشکال این است گاهی اوقات مدیران خودکار حافظه نمی‌توانند مکان‌های حافظه استفاده‌نشده را تخصیص دهند که ممکن است منجر به مشکل نشت حافظه شود. همچنین مقدار زیادی از زمان پردازنده را مصرف می‌کند و عملکرد غیرقطعی دارد [9].

۴-۷- زمان‌بندی

سیستم عامل بی‌درنگ به برنامه‌های بی‌درنگ امکان می‌دهد از طریق زمان‌بندی به ضرب‌الاجل خود برسند، رویکرد زمان‌بندی قلب تپنده هر سیستم بی‌درنگ است که اجرای هر کار را برای جلوگیری از همپوشانی کنترل می‌کند. سیستم‌های عامل بی‌درنگ برای عملکرد بسیاری از فناوری‌ها و فناوری‌هایی که برای زندگی روزمره ما حیاتی هستند، اساسی هستند. علاوه بر این، این سیستم‌ها ظرفیت اجرای نرم‌افزارهای لازم و ارائه دقت زمانی لازم را دارند، به این معنی که می‌توانند اطمینان حاصل بدهند که خاتمه یک برنامه یا اجرای کار مربوط به زمانی است که سیستم عامل بی‌درنگ برنامه‌ها را فعال می‌کند. از طریق زمان‌بندی به ضرب‌الاجل خود می‌رسند، رویکرد زمان‌بندی، قلب تپنده هر سیستم بی‌درنگی است که اجرای هر کار را برای جلوگیری از همپوشانی کنترل می‌کند [12].

طبقه‌بندی الگوریتم‌های زمان‌بندی بی‌درنگ بر اساس معیارهای مختلفی صورت می‌گیرد. این الگوریتم‌ها بر اساس زمان اولویت‌بندی وظایف به دو دسته تقسیم می‌شوند. اگر اولویت وظایف را قبل از اجرا مشخص نمایند الگوریتم‌های ایستا می‌نامند دارند؛ اما اگر در حین اجرا وظایف را اولویت‌بندی نمایند الگوریتم‌های پویا نامیده می‌شوند. این الگوریتم‌ها بر اساس اینکه در چه نوع محیطی مورد استفاده قرار گیرند به سه دسته الگوریتم‌های زمان‌بندی تک پردازنده، چند پردازنده متمرکز و توزیع شده تقسیم می‌شود [13].

^۱Automatic memory management

نحوه انجام وظایف که عموماً بر اساس اولویت است را زمان‌بندی^۱ می‌گویند. در این بخش، به طور خلاصه الگوریتم‌های زمان‌بندی مورد استفاده در سیستم‌عامل‌های بی‌درنگ را بیان می‌کنیم. توجه می‌کنیم که پیش‌بینی‌پذیری به اصول اولیه سیستم‌عامل محدود نیاز دارد. تحلیل امکان‌سنجی برنامه ممکن است در برخی موارد امکان‌پذیر نباشد. زمان‌بندی کار می‌تواند به صورت پیش‌گیرانه یا غیر پیش‌گیرانه و به صورت ایستا یا پویا انجام شود. برای برنامه‌های کوچک، زمان اجرای کار را می‌توان قبل از اجرا تخمین زد و زمان‌بندی وظایف اولیه را به صورت ایستا تعیین کرد. دو محدودیت رایج در زمان‌بندی، نیاز به منابع و تقدم اجرای وظایف است. هدف از برنامه‌ریزی، به حداقل رساندن یا حداکثر کردن اهداف معین است. نوعی از اهداف که به حداقل می‌رسند عبارت‌اند از: طول برنامه، متوسط تأخیر یا سستی. از طرف دیگر، به حداکثر رساندن میانگین زودهنگام و تعداد ورودی‌هایی که با موعد مقرر مطابقت دارند، می‌تواند هدف باشد. ما به طور خلاصه در مورد رویکردهای زمان‌بندی زیر صحبت می‌کنیم

رویکرد مبتنی بر جدول ایستا:

این الگوریتم‌ها معمولاً یک تحلیل ایستا مرتبط با زمان‌بندی را انجام می‌دهند و برنامه‌هایی را که سودمند هستند، ثبت می‌کنند. این به ارائه یک برنامه زمانی کمک می‌کند که می‌تواند وظیفه‌ای را که باید با آن اجرا در زمان اجرا شروع شود، مشخص کند [14].

امکان‌سنجی و برنامه‌ریزی به صورت ایستا تعیین می‌شود. یک مثال رایج، اجرایی دوره‌ای است که در بسیاری از سیستم‌های بی‌درنگ پویا در مقیاس بزرگ نیز استفاده می‌شود. وظایف را به شکاف‌های زمانی دوره‌ای اختصاص می‌دهد. در هر بازه زمانی (دوره)، کارها بر اساس جدولی که ترتیب اجرای وظایف را لیست می‌کند، اجرا می‌شود. برای کارهای تناوبی، یک برنامه زمانی امکان‌پذیر وجود دارد، اگر و تنها در صورتی که یک برنامه زمانی امکان‌پذیر در کمترین مضرب مشترک دوره‌ها وجود داشته باشد. نقطه ضعف این رویکرد این است که آگاهی قبلی از حداکثر الزامات وظایف در هر چرخه ضروری است.

رویکرد پیشگیرانه مبتنی بر اولویت ایستا:

مشابه رویکرد اول، این نوع الگوریتم‌ها از تحلیل ایستای زمان‌بندی نیز استفاده می‌کنند. تفاوت این است که به جای انتخاب یک برنامه زمانی خاص، روش مفیدی برای تعیین اولویت‌ها در بین وظایف مختلف در برنامه‌ریزی پیشگیرانه ارائه می‌دهد.

تحلیل امکان‌سنجی به صورت ایستا انجام می‌شود. وظایف بر اساس اولویت‌ها به صورت پویا از سال می‌شوند.

^۱Scheduling

^۲Deadline

متداول‌ترین الگوریتم زمان‌بندی پیشگیرانه مبتنی بر اولویت ایستا برای کارهای دوره‌ای، الگوریتم زمان‌بندی نرخ یکنواخت است [15]. یک سیستم تناوبی باید با یک خروجی قبل از ورودی بعدی پاسخ دهد. بنابراین زمان پاسخگویی سیستم باید کمتر از حداقل زمان بین ورودی‌های متوالی باشد. در الگوریتم زمان‌بندی نرخ یکنواخت به هر پردازش اولیتهی متناسب با فرکانس رخداد آن واقعه نسبت داده می‌شود و دوره تناوب کمتری داشته باشد. مثلاً به پردازشی که هر ۲۰ میلی ثانیه تکرار می‌شود، اولویت ۵۰ و به پردازشی که هر ۱۰۰ میلی ثانیه تکرار می‌شود، اولویت ۱۰ داده می‌شود. این الگوریتم از نوع غیرانحصاری است. می‌توان ثابت کرد که این الگوریتم بهینه است.

الگوریتم زمان‌بندی نرخ یکنواخت اولویت‌ها را متناسب با فراوانی وظایف تعیین می‌کند. اگر کل استفاده از منابع کمتر از $\ln 2$ باشد، می‌تواند هر مجموعه‌ای از کارها را برای رسیدن به مهلت‌های زمانی برنامه‌ریزی کند. اگر نتواند یک برنامه زمانی پیدا کند، هیچ طرح زمان‌بندی با اولویت ثابت دیگری این کار را نخواهد کرد. اما هیچ پشتیبانی از دوره‌ها/اولویت‌ها و وارونگی اولویت‌ها در حال تغییر را ارائه نمی‌کند. همچنین، وارونگی اولویت ممکن است زمانی رخ دهد که نرخ یکنواختی اعمال شود، یک کار غیر بحرانی فرکانس اجرا بالاتر است.

رویکرد مبتنی بر برنامه‌ریزی پویا:

زمان‌بندی‌های مبتنی بر برنامه‌ریزی پویا بر انجام بررسی‌های امکان‌سنجی به صورت پویا تمرکز می‌کنند. یک کار با ساختن یک طرح برای اجرای وظیفه تضمین می‌شود که به‌موجب آن تمام وظایف تضمین شده محدودیت‌های زمان‌بندی خود را برآورده می‌کنند [16]. یک کار با مجموعه‌ای از مفروضات تضمین می‌شود، به‌عنوان مثال، در مورد بدترین حالت زمان اجرا و نیاز به منابع، و ماهیت خطاها در سیستم. اگر این مفروضات برقرار باشند، پس از تضمین یک کار، الزامات زمان‌بندی آن را برآورده می‌کند. یک الگوریتم ضمانت باید مسائل زیادی از جمله زمان‌های اجرا در بدترین حالت، منابع موردنیاز، محدودیت‌های زمان‌بندی، وجود وظایف دوره‌ای، وظایف قابل پیش‌گیری، محدودیت‌های اولویت (که برای رسیدگی به گروه‌های وظیفه استفاده می‌شود)، سطوح اهمیت چندگانه برای وظایف، و الزامات تحمل خطا را در نظر بگیرد. در یک سیستم توزیع شده، زمانی که یک کار به یک سایت می‌رسد، زمان‌بندی‌کننده در آن سایت تلاش می‌کند تضمین کند که کار قبل از مهلت آن در آن سایت، اجرا می‌شود. اگر تلاش ناموفق باشد، مؤلفه‌های زمان‌بندی در سایت‌های جداگانه با یکدیگر همکاری می‌کنند تا تعیین کنند کدام سایت دیگر در سیستم منبع کافی برای تضمین کار دارد.

یک کار از یک منبع در حالت اشتراکی یا انحصاری استفاده می‌کند و منبع درخواستی را تا زمانی که اجرا می‌کند نگه می‌دارد

در اینجا، زمان‌بندی‌های امکان‌پذیر به صورت پویا (در زمان اجرا) شناسایی می‌شوند. این یک بازه زمانی مشخص را

حمل می‌کند و یک فرایند اجرا می‌شود اگر و تنها در صورتی که محدودیت زمانی را برآورده کند.

تجزیه و تحلیل امکان‌سنجی به صورت پویا انجام می‌شود - یک کار وارده برای اجرا فقط در صورت امکان پذیرفته می‌شود. تجزیه و تحلیل امکان‌سنجی نیز منبعی برای زمان‌بندی است. اجرای یک کار با دانستن بدترین زمان اجرای آن و خطاها در سیستم تضمین می‌شود. وظایف با واسطه‌گری منابع به صورت متمرکز یا از طریق مناقصه به سایت‌ها ارسال می‌شوند. یک تکنیک استفاده از رویکرد متمرکز و مناقصه نسبت به هر یک از آنها بسیار بهتر عمل می‌کند، اما پیچیده‌تر است.

رویکرد بهترین تلاش پویا:

این نوع رویکردها به جای زمان‌بندی‌های قابل اجرا، ضرب‌الاجل‌ها را در نظر می‌گیرند؛ بنابراین در صورت رسیدن به مهلت آن، تکلیف لغو می‌شود. این رویکرد به طور گسترده در اکثر سیستم‌های زمان واقعی استفاده می‌شود [17]. در اینجا هیچ بررسی امکان‌سنجی انجام نمی‌شود. بهترین تلاش برای رعایت ضرب‌الاجل‌ها و انجام وظایف ممکن است لغو شود. زمانی که اضافه‌بار وجود ندارد، بهینه هستند. تحقیقات در مورد شرایط اضافه‌بار هنوز در مراحل اولیه است. زمان‌بندی اولین مهلت اولیه می‌تواند سیستم‌های بی‌درنگ ایستا و پویا را برنامه‌ریزی کند. تجزیه و تحلیل امکان‌سنجی برای زمان‌بندی اولین مهلت اولیه را می‌توان در زمان $O(n^2)$ انجام داد، جایی که n تعداد وظایف است. برخلاف زمان‌بندی اولین مهلت اولیه، MLF زمان‌های اجرای کار را محاسبه می‌کند.

برنامه‌ریزی با تحمل خطا:

یک برنامه اولیه در مهلت مقرر در صورت عدم وجود خرابی اجرا می‌شود و یک برنامه ثانویه تا آخرین مهلت شکست اجرا می‌شود. چنین تکنیکی امکان تخریب برازنده را فراهم می‌کند؛ اما هزینه اجرای برنامه دیگری را متحمل می‌شود. در سیستم‌های بی‌درنگ سخت، مسدود کردن بدترین حالت باید برای تحمل خطا به حداقل برسد.

برنامه‌ریزی با بازیابی منابع:

زمان واقعی اجرای کار ممکن است کوتاه‌تر از زمانی باشد که قبلاً تعیین می‌شود، زیرا به دلیل مشروط‌ها یا مفروضات اجرای بدترین حالت است. توزیع‌کننده وظایف ممکن است سعی کند چنین کسری‌ها را به نفع وظایف غیر هم‌زمان یا تضمین‌های به موقع بهبود یافته، بازیابی کند.

۴-۸- سmafور

چندین رشته اجرا هم‌زمان در یک برنامه باید بتوانند اجرای خود را همگام کنند و دسترسی متقابل انحصاری به

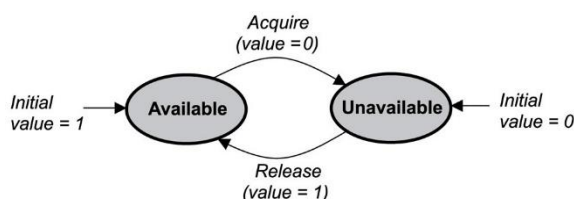
^۱EDF

منابع مشترک را هماهنگ کنند. سمافور^۱ یعنی توانایی همگام‌سازی دسترسی به منابع توسط چندین فرایند [18].

گاهی برای پیاده‌سازی ویژگی‌هایی نیاز است که چند وظیفه^۲ با هماهنگی با یکدیگر کار کنند. اینجاست که با مفهوم همگام‌سازی در سیستم‌عامل آشنا می‌شویم. یکی از ابزارهایی که سیستم‌عامل برای همگام‌سازی بین وظیفه‌ها و یا روتین‌های وقفه و وظیفه‌ها در اختیار ما می‌گذارد، سمافورها هستند که عملیات سیگنال دادن بین وظیفه‌ها را مدیریت می‌کنند. سمافورها به سه دسته تقسیم می‌شوند، سمافورهای باینری، شمارش کننده، و سمافورهای حذف متقابل.

سمافورهای باینری:

یک سمافور باینری^۳ می‌تواند مقدار ۰ یا ۱ داشته باشد.

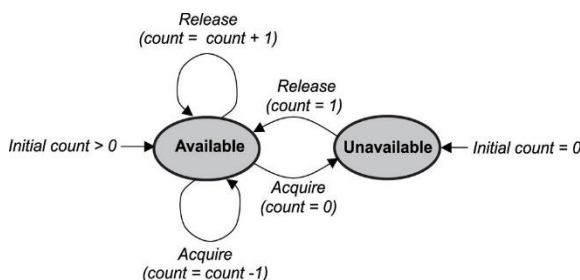


شکل ۳- ساختار سمافور باینری

طبق شکل ۳ وقتی مقدار یک سمافور باینری ۰ باشد، سمافور غیرقابل دسترسی (خالی) در نظر گرفته می‌شود. وقتی مقدار ۱ باشد، سمافور باینری در دسترس (کامل) در نظر گرفته می‌شود. توجه داشته باشید که وقتی یک سمافور باینری برای اولین بار ایجاد می‌شود، می‌توان آن را به صورت موجود یا غیرقابل دسترسی (به ترتیب ۱ یا ۰) مقداردهی اولیه کرد. نمودار حالت یک سمافور باینری در شکل نشان داده شده است.

سمافور شمارش کننده:

سمافور شمارشگر^۴ از یک شمارش استفاده می‌کند تا بتواند چندین بار به دست بیاید یا منتشر شود.



^۱Semaphore

^۲Task

^۳Including binary

^۴Counting

شکل ۴- ساختار سمافور شمارش‌کننده

طبق شکل ۴ هنگام ایجاد یک سمافور شمارش، به سمافور شمارشی اختصاص دهید که نشان‌دهنده تعداد نشانه‌های سمافور در ابتدا باشد. اگر تعداد اولیه ۰ باشد، سمافور شمارش در حالت غیرقابل دسترسی ایجاد می‌شود. یک یا چند کار می‌توانند به گرفتن یک نشانه از سمافور شمارش ادامه دهند تا زمانی که هیچ نشانه‌ای باقی نماند. هنگامی که تمام نشانه‌ها از بین رفتند، شمارش برابر با ۰ است و سمافور شمارش از حالت موجود به حالت غیرقابل دسترسی حرکت می‌کند.

سمافور محدودیت متقابل:

سمافور محدودیت متقابل الگوریتمی است که در برنامه‌نویسی همروند برای جلوگیری از استفاده هم‌زمان از منابع مشترک مانند متغیرهای سراسری در واقع برای دسترسی به یک منبع در یک‌زمان تنها به یک فرایند استفاده می‌شود. شیء *Mutex* به همه فرایندها اجازه می‌دهد تا از یک منبع استفاده کنند، اما در یک‌زمان، تنها یک فرایند مجاز به استفاده از منبع است. *Mutex* از تکنیک مبتنی بر قفل برای رسیدگی به مشکل بخش بحرانی استفاده می‌کند.

هر زمان که فرایندی منبعی را از سیستم درخواست کند، سیستم یک شیء *Mutex* با نام یا شناسه منحصر به فرد ایجاد می‌کند؛ بنابراین، هر زمان که فرایند بخواهد از آن منبع استفاده کند، آنگاه فرایند یک قفل روی شیء را اشغال می‌کند. پس از قفل کردن، فرایند از منبع استفاده می‌کند و در نهایت شیء *Mutex* را آزاد می‌کند. پس از آن، فرایندهای دیگر می‌توانند شیء *Mutex* را به همان شیوه ایجاد کرده و از آن استفاده کنند.

با قفل کردن شیء، آن منبع خاص به آن فرایند خاص تخصیص داده می‌شود و هیچ فرایند دیگری نمی‌تواند آن منبع را بگیرد؛ بنابراین، در بخش بحرانی، هیچ فرایند دیگری اجازه استفاده از منبع مشترک را ندارد. به این ترتیب، همگام‌سازی فرایند را می‌توان با کمک یک شیء *Mutex* به دست آورد [19].

۹-۳. امنیت در سیستم‌عامل بی‌درنگ

امنیت در سیستم‌عامل بی‌درنگ به این معنی است که در سیستم‌عامل بی‌درنگ احتمال به خطر انداختن جنبه امنیتی برای دستیابی به کارایی عملکرد بهتر وجود دارد. این مصالحه در امنیت ممکن است منجر به فاجعه شود؛ زیرا هرگونه نقض امنیت می‌تواند منجر به خطاهای عملیاتی شود. اغلب در هنگام طراحی سیستم تعبیه شده، هدف ارائه عملکرد خوب است و در فرایند تلاش برای به حداکثر رساندن عملکرد، به جنبه امنیتی در الزامات طراحی اهمیت لازم داده نمی‌شود. اگر سیستم‌عامل با در نظر گرفتن امنیت ساخته نشده باشد، همیشه در برابر حمله آسیب‌پذیر

^۱Mutual Exclusion

^۲Real-Time Operating System Security

خواهد بود و در جلوگیری و محدود کردن آسیب ناشی از دسترسی غیرمجاز شکست خواهد خورد. مقاومت در برابر چنین حملاتی تنها در صورتی می‌تواند تضمین شود که امنیت در معماری و پیاده سازی سیستم تعبیه شده باشد. هنگامی که مهاجمان به یک سیستم تعبیه شده دسترسی غیرمجاز پیدا می‌کنند، این پتانسیل را دارند که سیستم را به روش‌های مختلف خراب کنند. بسته به سیستمی که هدف قرار گرفته است، این حملات می‌تواند خسارات مالی زیادی و یا حتی تلفات جانی به همراه داشته باشند. این حملات می‌تواند یکپارچگی، محرمانه بودن، اصالت یا در دسترس بودن سیستم را به خطر بیندازند. موضوعات مطرح در امنیت سیستم عامل بی‌درنگ شامل ۶ مورد زیر است که هر یک را به اختصار توضیح می‌دهیم و پس از آن رویکردهای مقابله با آن را بیان می‌کنیم [20].

۴-۹-۱. سامانه نظارت، کنترل و جمع‌آوری داده‌ها ناامن

سیستم عامل بی‌درنگ در عصر مدرن به ندرت به تنهایی عمل می‌کند. معمولاً ماشین‌های خودکار خاصی را کنترل می‌کند و اغلب به یک سیستم نظارتی متصل می‌شود که برای مدیریت کل یک مرکز به سایر سیستم عامل بی‌درنگ متصل می‌شود. این شبکه‌ها اغلب به عنوان سامانه نظارت، کنترل و جمع‌آوری داده‌ها یا سیستم کنترل توزیع شده^۲ نامیده می‌شوند؛ بنابراین، امنیت ارتباطات درون سیستم عامل بی‌درنگ و بین سیستم عامل بی‌درنگ از اهمیت بالایی برخوردار است. امروز در همه جا حضور دارد. مردم آن را نمی‌بینند؛ زیرا شفاف یا فراگیر در زندگی روزمره آنهاست. معمولاً توسط برخی از شرکت‌ها یا کارمندان دولتی که مردم آن را نمی‌بینند، نظارت می‌شود. این شامل:

- برقی که به خانه شما می‌رود
- پمپ‌های آبی که آب را به خانه شما پمپاژ می‌کند
- آنچه از خانه شما بیرون می‌رود
- چراغ راهنمایی و رانندگی
- حمل و نقل عمومی
- تهویه مطبوع در ساختمانی که در آن کار می‌کنید
- تلفن همراهی که استفاده می‌کنید
- گاز طبیعی که به خانه شما می‌رود

۴-۹-۲. تزریق کد

حمله به سیستم هدف از طریق تزریق کد^۳ یکی از رایج‌ترین و بالقوه خطرناک‌ترین انواع حمله به سیستم‌های

^۱ سیستمی را در نظر بگیرید که قرار است در یک منطقه یا فضای صنعتی، کلیه فرآیندهای صنعتی را در همان مکان و یا از راه دور کنترل کند. این سیستم می‌خواهد کلیه داده‌های سنسورها، ولوها، محرکها و تجهیزات صنعتی را به شکل بلادرنگ جمع‌آوری کرده، آنها را مانیتور نموده و بر اساس پارامترهای تعریف شده در خود، تصمیمات خاصی را برای کنترل صحیح کلیه بخشها بگیرد. چنین سیستمی به عنوان سامانه نظارت، کنترل و جمع‌آوری داده‌ها (SCADA) شناخته می‌شود.

^۲DCS

^۳Code Injection

کامپیوتری مدرن است. چنین حملاتی با درج و اجرای کد دلخواه بر روی ماشین‌های قربانی کار می‌کنند. به‌طور کلی هدف کنترل شمارنده برنامه‌دستگاه است. این امر مهاجم را قادر می‌سازد تا جریان اجرای دستورالعمل‌ها را تغییر دهد و باعث شود رایانه شخصی کدهای مخرب وارد شده توسط خود مهاجم را اجرا کند. این کد تزریق شده می‌تواند کد منبع برای یک زبان برنامه‌نویسی تفسیر شده، کد بایت متوسط یا کد ماشینی قابل اجرا می‌باشد. چنین کدی ممکن است برای حمله فوری طراحی شده باشد، جایی که بلافاصله اجرا شود، یا ممکن است در زمان دیگری اجرا شود.

یکی دیگر از روش‌های متداول که مهاجمان سیستم‌ها را هدف قرار می‌دهند، بهره‌برداری از کدی است که ورودی یک منبع را می‌پذیرد. ورودی می‌تواند یک رشته ساده از کاراکترها باشد. اما اگر برای مسائلی مانند سرریز پشته و پشته بررسی نشود، می‌تواند باعث آسیب گسترده شود. مواردی وجود داشته است که سرریز بافر توسط تصاویر یا فایل‌های صوتی تعبیه شده به طور مخربی ایجاد شده است که سیستم‌عامل موفق به تأیید آن‌ها نشده و مهاجم را قادر می‌سازد تا کد دلخواه را روی سیستم اجرا کند.

۴-۹-۳. ناکارآمدی رمزگذاری

ناکارآمدی رمزگذاری به این معنا است که روش‌های زیادی برای رمزگذاری داده‌های عبوری از یک شبکه، چه در یک شرکت یا از طریق اینترنت وجود دارد. یکی از راه‌های محافظت از داده‌هایی که از کانال‌های ناامن عبور می‌کنند، استفاده از کلیدهای ترکیبی عمومی و خصوصی است. روش رمزگذاری کلید عمومی از دو کلید مرتبط ریاضی استفاده می‌کند. داده‌ها با یک کلید رمزگذاری می‌شوند و با کلید دیگر رمزگشایی می‌شوند؛ بنابراین یک فرایند نامتقارن است. اگر شخصی پیامی را ارسال کند که با استفاده از کلید عمومی رمزگذاری شده است، تنها شخصی که دارای خصوصی است می‌تواند پیام معتبر را رمزگشایی کرده و بخواند. از طرف دیگر، رمزگذاری کلید خصوصی از یک کلید استفاده می‌کند. این کلید فقط برای دو طرف درگیر در معامله شناخته شده است. این فرایند از نظر محاسباتی فشرده‌تر است، اما مستلزم آن است که کلید خصوصی باید خصوصی بماند.

این روش‌ها ممکن است به‌عنوان تکنیکی برای حل مشکلات ما در مورد حساسیت به یک حمله خارجی پیشنهاد شوند. اما رمزگذاری به‌خودی‌خود اثبات احمقانه نیست. این امکان برای مهاجم وجود دارد که یک بسته رمزگذاری شده را رهگیری کند و محتویات آن را قبل از ارسال آن تغییر دهد. این منجر به داده‌های درهم می‌شود. تکنیک‌های دیگری (ترکیبی از آنها) مانند هش وجود دارد که برای ایمن‌سازی انتقال داده‌ها استفاده می‌شود. اما این روش‌های پیچیده برای اکثر سیستم‌های تعبیه‌شده مالیات می‌دهند. دلیل اصلی، توانایی محاسباتی بالای موردنیاز برای رمزگذاری است. پردازنده سریع و منابع حافظه بالای موردنیاز برای چنین فعالیت‌های کمکی برای بسیاری از

^۱Program counter

^۲Inefficiency of encryption

سیستم‌های تعبیه شده که دارای پردازنده و حافظه بسیار محدودی هستند، بسیار دشوار است. یکی دیگر از دلایل عدم استفاده از رمزگذاری در مثال زیر نشان داده شده است. یک کنسول بازی را در نظر بگیرید که به اینترنت دسترسی دارد. اگر سیستم اجازه دسترسی به بازی‌های قسمت سوم را بدهد، احتمالاً نمی‌توانیم از رمزگذاری کلید خصوصی استفاده کنیم. از سوی دیگر، رمزگذاری کلید عمومی، سیستم را برای حمله از سوی هرکسی که تظاهر به توسعه‌دهنده بازی می‌کند باز می‌گذارد.

۴-۹-۴. بهره‌برداری از حافظه مشترک

بهره‌برداری از حافظه مشترک^۱ به این معنا است که سیستم‌عامل‌های بی‌درنگ اولیه فاقد واحد مدیریت حافظه بودند. آنها از یک مدل حافظه مشترک مسطح برای ارتباطات بین فرایندهای استفاده کردند. این روش را می‌توان با این فرض توضیح داد که حافظه مشترک مانند یک تخته سفید در اتاقی پر از افراد است. مردم از تابلو برای ارسال پیام به یکدیگر استفاده می‌کنند. شخص A وارد می‌شود و برای شخص B روی تخته سفید پیام می‌نویسد. با این حال شخص دیگری می‌تواند بباید و این پیام را قبل از اینکه شخص B آن را بخواند اصلاح کند. اگر قرار بود این اتفاق در یک سیستم کامپیوتری بیافتد، تصور اینکه چگونه کل سیستم ممکن است از کار بیفتد دشوار نخواهد بود. این روش کار می‌کند اما نیاز به تماس‌های سیستمی و مدیریت دقیق دارد. اما واضح است که چگونه یک فرایند مخرب می‌تواند محتویات حافظه را به راحتی خراب کند.

۴-۹-۵. وارونگی اولویت

وارونگی الویت^۲ به این معنا است که امروزه سیستم‌عامل بی‌درنگ نیاز به پردازش کارهای زیادی دارد و هر یک از آنها ممکن است منابعی مانند حافظه یا I/O را به اشتراک بگذارند. منابع محدودی از این دست را نمی‌توان به همان شیوه‌ای که حافظه به اشتراک گذاشته می‌شود به اشتراک گذاشت. از آنجایی که برخی از فرایندها ذاتاً مهم‌تر از سایرین هستند، در اولویت قرار می‌گیرند، جایی که فرایندهای با اولویت بالاتر می‌توانند یک سوئیچ زمینه با یک فرایند با اولویت پایین‌تر را مجبور کنند. یکی دیگر از مشکلات حافظه مشترک این است که یک فرایند باید بتواند یک بلوک از حافظه را به طور کامل بنویسد و در حین انجام این کار نمی‌توان آن را قطع کرد. این بدان معنی است که سایر فرایندهایی که می‌خواهند از حافظه استفاده کنند نمی‌توانند این کار را انجام دهند در غیر این صورت باعث ناسازگاری حافظه می‌شود. این مفهوم، طرد متقابل نامیده می‌شود که در آن تنها یک فرایند می‌تواند در یک زمان از یک منبع استفاده کند. حذف متقابل و اولویت فرایند به ترتیب سعی در حل مشکل اشتراک داده و اشتراک پردازشگر دارد.

۴-۹-۶. حمله‌های انکار سرویس

^۱Exploiting Shared Memory

^۲Priority Inversion

راه‌های مختلفی وجود دارد که از طریق حمله‌های انکار سرویس که یک کد مخرب کارایی سیستم را از بین ببرد و یا از کار بیندازد. فرض کنید یک سیستم از دو برنامه کاربردی تشکیل شده است که هر کدام یک وظیفه را اجرا می‌کنند در شرایط ایده‌آل، این دو فرایند زمان CPU را به اشتراک می‌گذارند که هر کدام ۵۰٪ زمان می‌برد. اما به دلیل طراحی بد یا قصد مخرب، اگر یکی از برنامه‌ها ۱۰۰ کار اضافی ایجاد کند، ۱۰۱ کار زمان اشتراک‌گذاری با برنامه خوب وجود دارد؛ بنابراین برنامه خوب از دسترسی به CPU محروم می‌شود. به جای ۵۰ درصد، کمتر از ۱ درصد از CPU را دریافت می‌کند.

انکار سرویس زمانی اتفاق می‌افتد که یک قسمت از سیستم منابع سیستم را به گونه‌ای کند که سایر بخش‌های سیستم نتوانند به طور عادی کار کنند. به عنوان مثال، اگر یک برنامه در یک حلقه بی‌پایان هنگ کند، برنامه‌های دیگر از زمان CPU محروم می‌شوند. یا شاید اگر یک برنامه حافظه زیادی را تخصیص دهد، نیاز به حافظه برنامه‌های دیگر را کم می‌کند. انکار سرویس می‌تواند به دلیل برنامه‌نویسی بد یا دسترسی ویروس یا هکر به سیستم به صورت خارجی باشد.

رویکردها و راه‌حل‌های مشترک

از آنجایی که سیستم‌های تعبیه شده شباهت بیشتری به سیستم‌های کامپیوتری دارند، برخی از راه‌حل‌ها مشابه هستند، مشکلات منحصربه‌فردی برای سیستم‌های جاسازی شده و سامانه نظارت، کنترل و جمع‌آوری داده‌ها (SCADA) وجود دارد، زیرا نیازمندی‌های بیشتر آن‌ها مانند زمان کار، عملکردهای حیاتی، ضرب‌الاجل و مصرف انرژی است.

۴-۹-۷. اجرای سیاست‌های امنیتی

با اجرای سیاست‌های امنیتی و استفاده از سامانه نظارت، کنترل و جمع‌آوری داده‌ها سیستم‌ها می‌توانند ایمن‌تر شوند. ابزارهایی به صورت آنلاین برای بررسی آسیب‌پذیری‌ها وجود دارد.

از آنجایی که سامانه نظارت، کنترل و جمع‌آوری داده‌ها در حال حاضر اغلب از شبکه‌های کامپیوتری رایج استفاده می‌کنند، مشکلی در امن کردن سامانه نظارت، کنترل و جمع‌آوری داده‌ها به اندازه شبکه‌های کامپیوتری وجود ندارد. فروشندگان می‌توانند با الزام رویه‌های راه‌اندازی شامل تغییر اجباری رمز عبور و بازنشانی زمانی که مشتریان محصول خود را خریداری می‌کنند، کمک کنند. شرکت‌ها می‌توانند کارمندان را از وصل کردن درایوهای USB از یک رایانه به رایانه دیگر برای جلوگیری از انتشار ویروس منع کنند.

سیستم اسکادا همچنین باید حساب‌های تمام کارمندان را پیگیری کند و همه کارمندان باید حساب شخصی خود را داشته باشند. بدیهی است که شبکه‌های خصوصی باید خصوصی باشند و یک کامپیوتر هرگز نباید به چندین شبکه در یک شرکت

^۱Denial of Service Attacks

^۲Enforcement of Security Policies

دسترسی داشته باشد. در نهایت، حساب‌های کارمندان سابق باید فوراً از سیستم او حذف شود تا از تلافی‌جویی جلوگیری شود.

۴-۹-۸. سیاست‌های امنیتی پویا از طریق اکتشافی

سیاست‌های امنیتی پویا از طریق اکتشافی^۱ به این معنا است که برای جلوگیری از نقض امنیت حتی در سیستم‌های ایمن جدیدتر. یک شبکه اسکادا را می‌توان با درک زمینه محیط آن به صورت سفارشی تنظیم کرد. اگر سیستم اسکادا در یک محیط تولید گاز طبیعی مستقر شود با یک کامپیوتر می‌توان تمام حسگرها را ردیابی کرد و هرگونه فرمانی را که ثبات سیستم را به خطر می‌اندازد، مجاز ندانست. به عنوان مثال، رایانه‌ای که شیرها و مخازن مختلف را نظارت می‌کند، ارتباط بین شیرها و فشار مخزن را مشاهده می‌کند. اگر یک کارمند دستوری را وارد کند که می‌تواند باعث خرابی شود، کامپیوتری که مستقیماً مقادیر را کنترل می‌کند باید دستور را رد کند. سپس دستور نامعتبر باید در یک پایگاه داده مرکزی ثبت شود. پایگاه داده متمرکز باید دارای خط‌مشی‌های کاربری چند سطحی باشد که کارمندان را به سطوح دسترسی مختلف تقسیم کند. دسترسی سطح پایین یا فقط خواندنی است یا تغییرات جزئی در سیستم‌های غیر بحرانی. دسترسی بیشتر به سطوح بالاتر و بالاتر داده می‌شود. مشخصات دسترسی بستگی به محیطی دارد که در آن مستقر شده است.

۴-۹-۹. استفاده از سطوح امتیاز

استفاده از سطوح امتیاز در سیستم عامل بی‌درنگ ابزاری برای جلوگیری از تأثیر جدی برنامه‌های مخرب، بر عملکرد سیستم است. برنامه‌های قانونی معمولاً نیازی به تغییر هیچ‌گونه عملیات سطح سیستم ندارند، مانند غیرفعال کردن وقفه‌ها. سیستم عامل این نوع دسترسی در سطح سیستم را فقط برای برنامه‌های خاصی محفوظ می‌دارد. معمولاً، برنامه‌هایی هستند که در طول توالی بوت وجود دارند. سایر برنامه‌ها (مانند برنامه‌های کاربردی دالود شده از شبکه) امتیاز کافی برای تغییر عملکردهایی که به طور بالقوه می‌توانند سیستم را غیرفعال کنند، ندارند.

چنین مکانیزمی را می‌توان با استفاده از حلقه‌ها اجرا کرد. این حلقه‌ها سطوح مختلف امتیازات را در یک سیستم نشان می‌دهند و تصمیم می‌گیرند که برنامه‌ای که در یک سطح امتیاز خاص اجرا می‌شود یا نمی‌تواند انجام دهد. به طور معمول یک سیستم دارای حداقل دو حلقه است. حلقه داخلی حالت هسته است. کمترین حفاظت را دارد و می‌تواند به بیشتر منابع دسترسی داشته باشد. سیستم عامل در هنگام راه‌اندازی در این حالت اجرا می‌شود. حلقه بیرونی مربوط به فضای کاربر است و بیشتر برای برنامه‌ها استفاده می‌شود. حداکثر سطح حفاظت و حداقل دسترسی به منابع را دارد.

۴-۹-۱۰. سطوح امنیت چندگانه مستقل

^۱Dynamic Security Policies through Heuristics

^۲Use of Privilege Levels

سطوح امنیت چندگانه مستقل^۱ یک معماری است که رویکرد لایه‌ای به امنیت را تعریف می‌کند. برخلاف هسته‌های یکپارچه که تمام عملکردهای قابل‌اعتماد را برای یک سیستم‌عامل ایمن انجام می‌دهند، سطوح امنیت چندگانه مستقل مفهوم هسته جدا سازی را برای اجرای مجموعه‌ای از سیاست‌های امنیتی کاربردی که جریان اطلاعات را تنظیم می‌کند، معرفی می‌کند.

معماری سطوح امنیت چندگانه مستقل یک چارچوب رسمی قابل‌استفاده مجدد برای امنیت سیستم با اطمینان بالا در سیستم‌های تعبیه شده فراهم می‌کند. هسته جدا سازی لایه پایه سیستم است و فضاهای فرایند سطوح امنیت چندگانه مستقل مجزایی را ایجاد می‌کند و جداسازی داده‌ها و کنترل جریان اطلاعات را در یک ریزپردازنده واحد اعمال می‌کند.

مزایای هسته جدا سازی سطوح امنیت چندگانه مستقل این است که می‌توان برای هر شیء دانست که ورودی‌ها از کجا آمده‌اند و خروجی‌ها کجا می‌روند. داده‌های هر شیء خصوصی باقی می‌ماند، علاوه بر این، هر لایه ممکن است به طور جداگانه ارزیابی و افزایش یابد، بدون اینکه بر لایه‌های دیگر تأثیر بگذارد. هنگامی که هسته جدا سازی اثبات شد و ایمن شد، نرم‌افزارهای سطح بالاتر ایمن، مانند مکانیزم ارتباطی امن، وب سرور یا سیستم فایل، می‌توانند در بالا قرار بگیرند. سیستمی که بر اساس چنین مفهومی ساخته شده است نیاز به پشتیبانی سخت‌افزاری خاصی دارد. تعدادی از عوامل متعددی که قبل از بررسی چنین سیستمی نیاز به توجه دارند در زیر آورده شده است.

قدرت پردازش: سیستم بی‌درنگ باید قابلیت محاسبات کافی برای برآورده کردن بدترین نیاز زمان‌بندی سیستم را داشته باشد.

حالت ممتاز: برخی دستورالعمل‌ها فقط توسط هسته جدا سازی اجرا می‌شوند، بنابراین سیستم باید یک حالت ممتاز را فراهم کند.

واحد مدیریت حافظه: واحد مدیریت حافظه جداسازی فضاهای آدرس بین پارتیشن‌ها را فراهم می‌کند. جداسازی داده‌ها یا محدودیت آسیب بدون پشتیبانی سخت‌افزاری برای جدا سازی امکان‌پذیر نیست. پردازنده باید به منابع حافظه موردنیاز دسترسی داشته باشد و باید به هسته جدا سازی توانایی محدود کردن دسترسی پارتیشن به حافظه را بدهد.

تله‌های دستورالعمل: تله‌های دستورالعمل^۲ به این معنا است که اگر یک پارتیشن تلاش می‌کند یک عملیات ممتاز

^۱MILS

^۲Privileged Mode

^۳MMU

^۴Instruction Traps

یا نامعتبر را اجرا کند، پردازنده باید مکانیزمی برای انتقال کنترل به SK داشته باشد. اینها ویژگی‌های اصلی پردازنده هستند و به‌طور کلی در بسیاری از ریزپردازنده‌های تجاری موجود هستند.

به این ترتیب، می‌بینیم که از طریق جداسازی، می‌توانیم یک مدل سلسله‌مراتبی از خدمات امنیتی ایجاد کنیم. هر لایه از این مدل از خدمات امنیتی یک لایه پایین‌تر برای ارائه یک عملکرد امنیتی جدید استفاده می‌کند که می‌تواند توسط لایه‌های بالاتر استفاده شود. هر لایه تنها مسئول امنیت خود است. مفهوم هسته جداسازی قدرتمند است؛ زیرا به نرم‌افزارهایی با الزامات امنیتی مختلف اجازه می‌دهد تا روی یک پردازنده واحد اجرا شوند.

برای مثال، برنامه‌ای که حاوی داده‌ها و الگوریتم‌های طبقه‌بندی شده است می‌تواند با خیال راحت یک پارتیشن را اشغال کند در حالی که پارتیشن دیگر به اینترنت طبقه‌بندی نشده متصل است.

۵- تفاوت‌های سیستم‌عامل‌های بی‌درنگ با سیستم‌عامل‌های معمول

تفاوت سیستم‌عامل بی‌درنگ با سیستم‌عامل معمولی شامل موارد زیر است که هر یک را به اختصار توضیح می‌دهیم.

۵-۱. جبرگرایی

جبرگرایی^۱ به این معنا است که در سیستم‌عامل‌های بی‌درنگ همواره بیشترین مدت‌زمان اجرای هر وظیفه مشخص است. ولی در سیستم‌عامل‌های عمومی مدت‌زمان اجرای هر وظیفه مهم نیست.

۵-۲. مدیریت کارها

مدیریت کارها^۲ در سیستم‌عامل‌های عمومی طوری طراحی می‌شوند تا چندین وظیفه مختلف به صورت هم‌زمان اجرا شوند؛ لذا کلیه وظیفه‌ها بدون توجه به اولویت آنها اجرا می‌شوند؛ لذا ممکن است وظیفه‌ای با اولویت پایین به وظیفه‌ای با اولویت بالاتر مقدم شود. ولی در سیستم‌عامل‌های بی‌درنگ هرگاه یک وظیفه با اولویت بالاتر نیاز به سخت‌افزار داشته باشد، وظیفه در حال اجرا متوقف شده و وظیفه با اولویت بالاتر کنترل سخت‌افزار به عهده می‌گیرد. هر وظیفه در سیستم‌عامل‌های بی‌درنگ دارای مشخصه اولویت است طوریکه وقتی یک وظیفه با اولویت بالاتر نیاز به اجرا داشته باشد، وظیفه با اولویت پایین متوقف شده و وظیفه با اولویت بالاتر اجرا می‌شود. این امر موجب می‌شود که سیستم به رویدادهای مهم، سریع پاسخ دهد.

۵-۳. کاربرد

کاربرد^۳ سیستم‌عامل‌های بی‌درنگ غالباً در سیستم‌های تعبیه شده^۴ استفاده می‌شوند؛ ولی سیستم‌عامل‌های عمومی

^۱Determinism

^۲Task management

^۳Application

^۴ سیستم‌های تعبیه شده، سیستم‌های کامپیوتری با عملکردی مخصوص و هدفی مشخص هستند که درون سیستم‌های بزرگتر تعبیه می‌شوند

در کامپیوترهای شخصی و دیگر کامپیوترهایی که استفاده همه‌منظوره دارند، استفاده می‌شوند.

۴-۵. تأخیر

تأخیر در سیستم‌عامل‌های معمولی چندان مهم در نظر گرفته نمی‌شود. ولی تأخیر وقفه در سیستم‌عامل عامل بی‌درنگ حداقل در چند میکروثانیه اندازه‌گیری می‌شود.

۵-۵. هسته پیشگیرانه

هسته پیشگیرانه نوعی از هسته است که همیشه بالاترین اولویت را که آماده اجرا هستند، اجرا می‌کند. در حالی که در سیستم‌عامل‌های معمولی نوعی از هسته است که از شرایط مسابقه در ساختارهای داده هسته آزاد است

۶- محبوب‌ترین سیستم‌عامل‌های بی‌درنگ در سال ۲۰۲۱

محبوب‌ترین سیستم‌عامل‌های بی‌درنگ در سال ۲۰۲۱ در زیر لیست شده است که هر یک را به‌اختصار توضیح می‌دهیم.

▪ Deos

این سیستم‌عامل مربوط به یک شرکت خصوصی است که ابتدا در دانمارک تأسیس شد، کاربردهای حیاتی در حوزه‌های صنعتی، خودرو، سیستم‌های بدون سرنشین، صنعت دفاعی، لپ‌تاپ‌های ایمن، زیرساخت‌های حیاتی و برای بسیاری از عملکردهای حیاتی پرواز که به پردازش محدود، جبر بالا و توان عملیاتی بالا نیاز دارند، استفاده شده است. Deos از زمان اولین تأیید توسط Transport Canada در سال ۱۹۹۸، به‌عنوان یک سیستم‌عامل بی‌درنگ حیاتی از نظر ایمنی ثابت شده است، و گواهی‌نامه دریافت کرده و در ۱۰ هزار هواپیما در حال پرواز است.

▪ embOS

embOS یک سیستم‌عامل بی‌درنگ با اولویت کنترل است embOS. برای همه هسته‌ها، کامپایلرها و ابزارهای توسعه محبوب در دسترس است و در چندین میلیارد دستگاه در طیف وسیعی از زمینه‌های کاربردی مستقر شده است.

embOS برای هرگونه استفاده غیرتجاری مانند آموزش و ارزیابی، بدون هیچ‌گونه محدودیت فنی رایگان است. مدل تجاری بدون حق امتیاز است، با مشتری برای دسترسی به کد منبع برای حقوق توزیع در سیستم‌های تولید. تمرکز بر صنعت و خودروسازی است، با گواهی‌نامه‌هایی برای استانداردهای تضمین ایمنی مناسب در آن مناطق.

^۱Latency

^۲Preemptive kernel

^۳www.ddci.com/products_deos_do_178c_arinc_653

^۴<https://www.segger.com/products/rtos/embos>

▪ FreeRTOS

تصور می‌شود که^۱ FreeRTOS محبوب‌ترین سیستم‌عامل بی‌درنگ برای طراحی‌ها باشد. از طیف متنوعی از معماری‌های پردازنده پشتیبانی می‌کند و خرید آن توسط آمازون باعث افزایش سرمایه‌گذاری در مهندسی شده است.

FreeRTOS یک سیستم‌عامل متن‌باز و بی‌درنگ برای میکروکنترلرها است که برنامه‌ریزی، استقرار، ایمن سازی، اتصال و مدیریت دستگاه‌های کوچک و کم‌مصرف را آسان می‌کند. شامل یک هسته و مجموعه در حال رشدی از کتابخانه‌های نرم‌افزاری است که برای استفاده در بخش‌های صنعتی و برنامه‌های کاربردی مناسب است. FreeRTOS با تأکید بر قابلیت اطمینان و سهولت استفاده ساخته شده است و قابلیت پیش‌بینی نسخه‌های پشتیبانی طولانی مدت را ارائه می‌دهد.

▪ Integrity

حول یک معماری پارتیشن‌بندی ساخته شده است که به توسعه‌دهندگان این امکان را می‌دهد تا اطمینان حاصل کنند که برنامه‌هایشان بالاترین نیازهای ممکن را برای امنیت، قابلیت اطمینان و عملکرد برآورده می‌کنند. INTEGRITY با شجره رهبری خود که توسط طیف وسیعی از گواهی‌نامه‌ها تأکید شده است، استانداردهایی را برای ایمنی، امنیت و قابلیت اطمینان سیستم‌عامل بی‌درنگ تعیین می‌کند. INTEGRITY از محافظت از حافظه سخت‌افزاری برای جدا سازی و محافظت از برنامه‌های تعبیه شده استفاده می‌کند. پارتیشن‌های امن منابعی را که برای اجرای صحیح هر کار نیاز دارد تضمین می‌کنند و به طور کامل از سیستم‌عامل و وظایف کاربر در برابر کدهای اشتباه و مخرب محافظت می‌کند - از جمله حملات انکار سرویس، کرم‌ها و اسب‌های تروjan.

برخلاف سایر سیستم‌عامل‌های محافظت شده با حافظه، INTEGRITY هرگز عملکرد بی‌درنگ را قربانی امنیت حفاظت نمی‌کند.

معماری INTEGRITY از چندین فضای آدرس مجازی محافظت شده پشتیبانی می‌کند که هر کدام می‌توانند چندین کار کاربردی داشته باشند.

▪ Keil RTX

Keil RTX یک سیستم‌عامل بی‌درنگ و قطعی و بدون حق امتیاز است که برای دستگاه‌های ARM و Cortex-M طراحی شده است. این سیستم‌عامل بی‌درنگ به شما امکان می‌دهد برنامه‌هایی ایجاد کنید که به طور هم‌زمان چندین عملکرد را انجام دهند و به ایجاد برنامه‌هایی که ساختار بهتری دارند و به راحتی نگهداری می‌شوند

^۱<https://aws.amazon.com/freertos>

^۲<https://www.ghs.com/products/rtos/integrity>

^۳<https://www.keil.com/arm/rl-arm/kernel>

کمک می‌کند.

▪ LynxOS

یک شرکت خصوصی است که در سال ۱۹۸۵ تأسیس شد (LinuxWorks سابق) و در سن خوزه، کالیفرنیا مستقر است. تمرکز بر بازارهای A&D برای مشتریانی که به گواهی DO-178 نیاز دارند، داشته و با گسترش روزافزون به زیرساخت‌های خودرو، صنعتی و فناوری اطلاعات روبه‌روز توسعه می‌یابد.

LynxOS-178 یک سیستم‌عامل بی‌درنگ است که از پلتفرم‌های x86، Arm و PowerPC پشتیبانی می‌کند. این سیستم‌عامل مبتنی بر استانداردهای باز است و به طور خاص برای برآورده کردن نیازهای سختگیرانه برنامه‌های کاربردی چندرشته‌ای و چند فرایندی در سیستم‌های بی‌درنگ حیاتی، ایجاد امنیت و ایمنی از طریق جداسازی سخت‌افزاری بین فرایندهای بی‌درنگ، کاربرد دارد.

▪ Nucleus

توسعه‌دهندگان سیستم را قادر می‌سازد تا نیازهای پیچیده موردنیاز طرح‌های تعبیه شده پیشرفته امروزی را برطرف کنند.^۲ Nucleus قابلیت‌های غنی از هسته و ویژگی‌های ابزاری ایده‌آل را برای برنامه‌هایی که در آن رد پای مقیاس‌پذیر، اتصال، امنیت، مدیریت انرژی و عملکرد قطعی ضروری هستند، گرد هم می‌آورد. Nucleus RTOS یک سیستم‌عامل بی‌درنگ اثبات شده، قابل اعتماد و کاملاً بهینه است. Nucleus با موفقیت در بازارهای بسیار پرتقاضا با الزامات ایمنی و امنیتی دقیق مانند سیستم‌های صنعتی، دستگاه‌های پزشکی، سیستم‌های هوابرد، خودرو و غیره مستقر شده است.

▪ Neutrino

از سال ۱۹۸۰، هزاران شرکت از سیستم‌عامل‌های بی‌درنگ QNX® استفاده کرده‌اند تا از ترکیب ایده‌آل عملکرد، امنیت و قابلیت اطمینان در سیستم‌های حیاتی اطمینان حاصل کنند. هسته فناوری QNX، سیستم‌عامل بی‌درنگ (RTOS) Neutrino® QNX® است، یک سیستم‌عامل بی‌درنگ کاملاً برجسته که محصولات نسل بعدی را در هر صنعتی که قابلیت اطمینان اهمیت دارد، از جمله خودرو، دستگاه‌های پزشکی، رباتیک، حمل‌ونقل و سیستم‌های تعبیه‌شده صنعتی قابل استفاده است.

▪ Azure RTOS ThreadX

Azure RTOS ThreadX یک سیستم‌عامل بی‌درنگ صنعتی پیشرفته مربوط به شرکت مایکروسافت است. Azure RTOS ThreadX برنامه‌ریزی پیشرفته، ارتباطات، همگام‌سازی، تایمر، مدیریت حافظه و امکانات مدیریت وقفه را

^۱<https://www.lynx.com/products/lynxos-178-do-178c-certified-posix-rtos?hsLang=en>

^۲<https://www.plm.automation.siemens.com/global/en/products/embedded/nucleus-rtos>

^۳<https://blackberry.qnx.com/en/products/foundation-software/qnx-rtos#top>

^۴<https://docs.microsoft.com/en-gb/azure/rtos/threadx/overview-threadx>

فراهم می‌کند. علاوه بر این، Azure RTOS ThreadX دارای بسیاری از ویژگی‌های پیشرفته است: از جمله معماری picokernel، برنامه‌ریزی preempton-thresholdTM، زنجیره رویداد، نمایه‌سازی، معیارهای عملکرد، و ردیابی رویداد سیستم. Azure RTOS ThreadX همراه با سهولت استفاده عالی، انتخاب ایده‌آلی برای سخت‌ترین برنامه‌های تعبیه شده است. Azure RTOS ThreadX میلیاردها استقرار در طیف گسترده‌ای از محصولات، از جمله دستگاه‌های مصرف‌کننده، الکترونیک پزشکی، و تجهیزات کنترل صنعتی دارد.

▪ WxWord

VxWorks در طیف وسیعی از بازارهای صنعتی و هوافضا وجود دارد. اخیراً یک شرکت امنیتی لینوکس، Starlab در سال ۲۰۲۰ خریداری کرده است.

VxWorks پراستفاده‌ترین سیستم عامل بی‌درنگ در صنعت سیستم‌های جاسازی شده است. نمونه‌هایی از برنامه‌هایی که از VxWorks استفاده می‌شود ایستگاه فضایی بین‌المللی و در روبات‌های مریخ‌نورد معروف ناسا Spirit و Opportunity است. این سیستم عامل بی‌درنگ توسط چندین آژانس و استانداردهای بین‌المللی برای ۲۴۴۸ سیستم بی‌درنگ، قابلیت اطمینان و برنامه‌های کاربردی حیاتی امنیتی تأیید شده است. هسته VxWorks اجازه می‌دهد تا وظایف بر اساس عملکرد گروه‌بندی شوند و در فرایندهای زمان اجرا قرار گیرند. هر فرایند زمان اجرا طوری طراحی شده است که وظایف را از یکدیگر جدا کند تا در صورت خرابی یکی از آنها، از سیستم محافظت کند.

¹RTP

- [1] P. Kampert, "A taxonomy of virtualization technologies," 2010.
- [2] A. K. Shukla, R. Sharma, P. K. Muhuri and , "A review of the scopes and challenges of the modern real-time operating systems," International Journal of Embedded and Real-Time Communication Systems (IJERTCS), vol. 9, no. 1, pp. 66-82, 2018.
- [3] I. Ungurean and N. C. Gaitan, "Performance analysis of tasks synchronization for real time operating systems," in 2018 International Conference on Development and Application Systems (DAS), 2018.
- [4] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," Communications of the ACM, vol. 17, pp. 412-421, 1974.
- [5] A. Desai, R. Oza, P. Sharma and B. Patel, "Hypervisor: A survey on concepts and taxonomy," International Journal of Innovative Technology and Exploring Engineering, vol. 2, pp. 222-225, 2013.
- [6] F. Rodríguez-Haro, F. Freitag, L. Navarro, E. Hernández-sánchez, N. Farías-Mendoza, J. A. Guerrero-Ibáñez and A. González-Potes, "A summary of virtualization techniques," vol. 3, pp. 267-272, 2012.
- [7] P. Kedia, R. Nagpal and T. P. Singh, "A survey on virtualization service providers, security issues, tools and future trends," International Journal of Computer Applications, vol. 69, 2013.
- [8] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," IEEE Transactions on Computers, vol. 100, pp. 933-940, 1987.
- [9] M. S. Johnstone and P. R. Wilson, "The memory fragmentation problem: Solved?," ACM Sigplan Notices, vol. 34, pp. 26-36, 1998.
- [10] S. Baskiyar and N. Meghanathan, "A Survey of Contemporary Real-time Operating Systems.," vol. 29, 2005.
- [11] R. L. Budzinski and E. S. Davidson, "A comparison of dynamic and static virtual memory allocation algorithms," IEEE Transactions on software Engineering, pp. 122-131, 1981.
- [12] I. GA, S. AA, A. AL, O. N, M. KJ, A. AJ, S. NO, H. SS, K. SF, I. IM and Y. HM., "Scheduling Algorithms Implementation for Real Time Operating Systems: A Review," Asian Journal of Research in Computer Science, vol. 11, no. 4, pp. 35-51, 2021.
- [13] Y. Lor Mohammed Hassani and M. Naghavi, "New approaches to scheduling in real-time systems," in 3rd National conference on Computer Engineering and IT Management, Tehran, 1395 (In Persian).
- [14] A. Vaishnav, K. D. Pham and D. Koch, "A Survey on FPGA Virtualization," in 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, 2018.
- [15] W. Wang, M. Bolic and J. Parri, "pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment," in IEEE, Montreal, 2013.

- [16] K. Rupnow, W. Fu and K. Compton, "Block, Drop or Roll(back): Alternative Preemption Methods for RH Multi-Tasking," in 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, CA, USA, 2009.
- [17] A. Bourge, O. Muller and F. Rousseau, "Generating Efficient Context-Switch Capable Circuits through Autonomous Design Flow," ACM, vol. 10, no. 1, pp. 1-23, 2017.
- [18] M. Asiatici, N. George, K. Vipin, S. A. Fahmy and P. Jenne, "Virtualized Execution Runtime for FPGA Accelerators in the Cloud," IEEE, vol. 5, no. 1, pp. 1900-1910, 2017.
- [19] C. Li, A. Raghunathan and N. k. Jha, "Secure Virtual Machine Execution under an Untrusted Management OS," in IEEE, Miami, FL, USA, 2010.
- [20] J. Smith and R. Nair, "The architecture of virtual machine," IEEE Compute Society, vol. 38, no. 5, pp. 32-38, 2005.
- [21] P. Kampert, A taxonomy of virtualization technologies, Delft: Delft University of Technology, 2010.
- [22] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," ACM, vol. 17, no. 7, pp. 412-421, 1974.
- [23] A. Desai, R. Oza, P. Sharma and B. Patel, "Hypervisor: aSurvey on Concepts and Taxonomy," International Journal of Innovative Technology and Exploring Engineering(IJITEE), vol. 2, no. 3, pp. 222-225, 2013.
- [24] F. Rodriguez-Haro, F. Freitag, L. Navarro, E. Hernandez-Sanchez, N. Frias-Mendoza, J. A. Guerrero-Ibanez and A. Gonzalez-Potes, "A Summary of virtualization techniques," Elsevier, vol. 3, no. 1, pp. 267-272, 2012.
- [25] P. Kedia, R. Nagpal and T. P. Singh, "A Survey on Virtualization Service Providers, Security Issues, Tools and Future Trends," International Journal of Computer Applications, vol. 69, no. 24, pp. 36-42, 2013.
- [26] W. v. Hagen, Professional XEN Virtulization, Indianapolis, IN, United States: John Wiley & Sons Inc, 2008.
- [27] B. Singh, J. Singh and S. Kumar, "Virtualization Techniques and Virtualization Challenges in Cloud Computing: A Review," IJCAT - International Journal of Computing and Technology, vol. 2, no. 6, pp. 175-180, 2015.
- [28] V. G. d. Silva, M. Kirikova and G. Alksnis, "Containers for Virtualization: An Overview," RIGA technical University, vol. 23, no. 1, pp. 21-27, 2018.
- [29] S. Nanda and T.-c. Chiueh, "A Survey on Virtualization Technologies," Department of Computer Science SUNY at Stony Brook, NewYork, 2005.
- [30] S. Baskiyar and N. Meghanathan, "A Survey of Contemporary Real-time Operating Systems.," 2005.

Provide a classification based on the characteristics of real-time operating systems

Mehdi Ghezlesflo, Mehdi Naghvi

Abstract

A real-time operating system is an operating system where time is the key factor. A real-time system is used when it is necessary to perform processing operations or data flow at a well-defined time. A real-time system is often used as a control tool in a specific application. The basic structure of the real-time operating system is similar to the normal operating systems, which, in addition to its characteristics, provides the possibility of time management of tasks with the help of methodologies such as synchronization, interrupt management, scheduling, memory management, etc., which leads to the wide use of the real-time operating system in flight control systems. Missile guidance systems, medical systems, nuclear reactor control systems, aircraft autopilot systems, anti-missile systems, and other control systems are sensitive. This type of operating system has a significant application in the industry; Therefore, compiling and presenting a classification according to the latest reviews and achievements in this field will be of great help to the thinkers of this field. In this article, while compiling and presenting this classification, there is a comprehensive review of real-time operating system and its topics including synchronization, interrupt management, real-time priority levels, memory management, scheduling, semaphore, fast job prepayment, security and the most used real-time operating systems.