دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی

نیمسال دوم ۱۴۰۰-۱۴۰۱

اصول سیستمهای عامل

جلسه ۱۷

نگارنده: زهرا رمضانی

۱ اردیبهشت ۱۴۰۱

فهرست مطالب

۱ خاتمه فرآیندها

۱.۱ خاتمه توسط کاربر/برنامه‌ی کاربردی خرابکار

به چنین خاتمه‌ای اصطلاحا kill کردن یک فرآیند اطلاق می‌گردد.

۲.۱ خاتمه فرآیند توسط فرآیند پدر

یک فرآیند ممکن است با استفاده از فراخوانهای سیستمی توسط فرآیند دیگری خاتمه یابد. به‌طور معمول چنین فرآیندهایی توسط فرآیند پدر (Parent) انجام می‌شود.

ضرورت آگاهی شناسه فرآیند فرزند توسط فرآیند پدر. یک فرآیند اگر بخواهد فرآیند فرزند خود را خاتمه دهد باید شناسه آن را بداند. بنابراین وقتی یک فرایند جدید به عنوان مثال با فراخوان سیستمی نظیر ()fork ایجاد میشود، شناسه آن فرآیند برای فرآیند پدرش ارسال میشود.

مطلب تکمیلی: خروجی حاصل از فراخوانی ()fork توسط یک فرآیند یک شناسه فرآیند (PID) است که میتواند مقادیر زیر را به خود بگیرد:

۱. PID:>۰ فرآیند جدید به درستی ایجاد شده و از این به بعد فرآیند با شناسه PID معتبر است،

۲. PID:<۰ شناسه نامعتبر بوده و فرآیند جدید باموفقیت ایجاد نشده است،

۳. PID:=۰ اگر سورس فرآیند پدر و فرزند یکسان باشد، مقدار خروجی دستور ()fork برای فرآیند فرزند حاوی مقدار صفر است. لازم به ذکر است که برای فرآیند پدر، خروجی تابع ()fork، PID فرآیند جدید است.

برخی از دلایل که فرآیند پدر سبب خاتمه فرزند می‌شود عبارتند از:

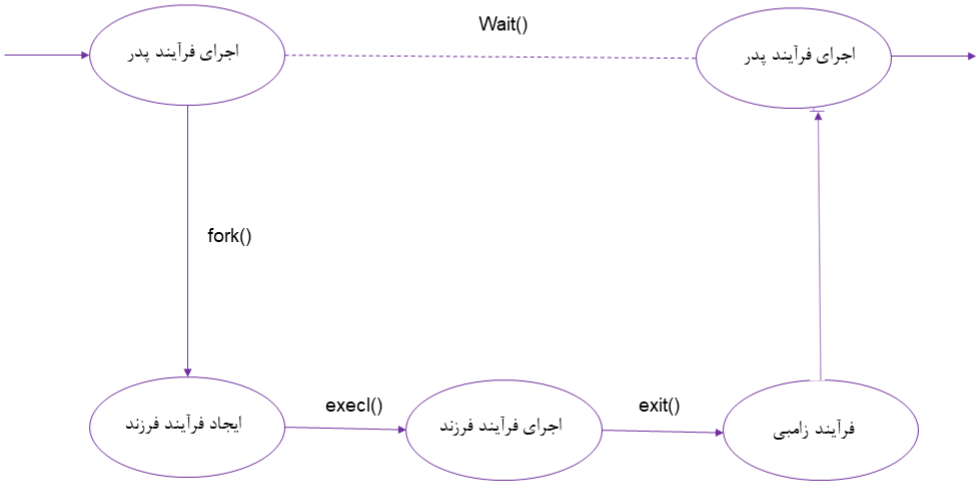
۱. استفاده بیش از حد فرآیند فرزند از برخی منابع تخصیصی،

۲. عدم نیاز به وظیفه محول شده به فرآیند فرزند،

۳. اتمام فرآیند پدر (به صورت نرمال/غیرنرمال) و عدم اجازه برخی سیستمها به ادامه حیات فرآیندهای فرزند.

خاتمه آبشاری فرآیندها (Cascading Termination). به نحوه خاتمه‌ای که در آن اتمام فرآیند پدر موجب خاتمه تمامی فرآیندهای فرزند می‌شود، خاتمه آبشاری (cascading termination) اطلاق می‌شود.

فرآیند زامبی(Zombie Process). هنگامی که یک فرآیند خاتمه پیدا میکند تا زمانی‌که خاتمه آن توسط فرآیند پدر شناسایی میشود، به فرآیند مذکور، فرآیند زامبی اطلاق میشود. در واقع با خاتمه فرآیند، منابع تخصیص یافته به آن آزاد میشود ولی همچنان چنین فرآیندی در جدول فرآیندها باقی میماند تا فرآیند پدر از وضعیت آن مطلع شود. شکل زیر مفاهیم بالا را به صورت مصور نشان میدهد.



سوال: فرض کنید فرآیند پدر بنا به هر دلیلی خاتمه یابد (شکل قبل را در نظر بگیرید). در چنین وضعیتی به طور خاص فرآیندهای زامبی مربوط به این فرآیند دیگر فاقد فرآیند پدر در سیستم هستند و اصطلاحا به آنها فرآیندهای یتیم (Orphan Process) اطلاق میشود. حال سوال این است که در چنین وضعیتی آزاد کردن مداخل جدول فرآیند و آزادسازی فرآیندهای یتیم بر عهده چه کسی است؟

پاسخ: در سیستمها به طور معمول چنین حالتی را با در نظر گرفتن فرآیند با PID=1 (فرایند Systemd) به عنوان پدر فرآیندهای یتیم کنترل و مدیریت میکنند. بدین نحو که فرآیند مذکور با فراخوانی تابع Wait به طور دورهای سبب آزادسازی فرآیندهای یتیم و آزاد کردن مداخل جدول فرآیندها میشود.

مطلب تکمیلی: دلایل خاتمه فرآیند و استراتژیهای مربوط به آن به صورت گستردهای در سیستمهای مختلف قابل ردهبندی هستند. به عنوان مثال در سیستمعاملهای اندروید برای فرآیندهای سلسله مراتبی از الویتها لحاظ میشود که بر اساس آن سعی میشود به منظور محدودیت حافظه، فرآیندهایی برای خاتمه و آزادسازی منابع انتخاب شود که الویت کمتری دارند. سلسله مراتب لحاظ شده به ترتیب از الویت بالا به الویت پایین به قرار زیر است:

۱. **فرآیندهای پیش زمینه (Foreground Process)**: فرآیندهای جاری قابل مشاهده روی صفحه که کاربر در حال حاضر با آنها تعامل دارد،
۲. **فرآیندهای قابل مشاهده (Visible Process)**: فرآیندهایی که به طور مستقیم روی پیشزمینه قابل مشاهده نیستند اما در حال انجام فعالیتی هستند که فرایندهای پیشزمینه به آنها ارجاع میکنند (مثلا فرآیندی که در حال انجام فعالیتی است و وضعیت آن روی صفحه نمایش داده میشود).
۳. **فرآیند سرویس (Service Process)**: شبیه فرآیندهای پسزمینه هستند با این تفاوت که در حال انجام فعالیتی هستند که قابل مشاهده توسط کاربر است (مثلا فرآیند پخش موسیقی).
۴. **فرآیندهای پسزمینه (Background Process)**: فرآیندهایی هستند که در حال انجام فعالیتی میباشند که از دید کاربر پنهان است.
۵. **فرآیند تهی (Empty Process)**: فرآیندی است که هیچگونه مولفه فعال (Active Component) تخصیص داده شده به برنامه کاربردی ندارد.

نکته: فرآیندها در چنین سیستمهایی ممکن است به طور همزمان چند ویژگی بالا را داشته باشند به عنوان مثال هم Visible باشند و هم Service باشند، در این صورت الویت در نظر گرفته شده برای آنها در نمونه مذکور بالاترین سطح (یعنی Visible) خواهد بود.

۲ ارتباط داخلی فرآیندها (IPC = Interprocess Communication)

فرآیندهای موجود در سیستم را میتوان از لحاظ همروندی به دو گروه تقسیمبندی کرد:

۱. **فرآیندهای مستقل (Independent)**: به فرآیندهایی اطلاق میشود که یا هیچ فرآیند در حال اجرای دیگری دادهای را به اشتراک نمیگذارند.
۲. **فرآیندهای همکار (Cooperate)**: هر فرآیندی که اشتراک گذاری داده با دیگر فرآیند همکار شناخته میشود. در یک تعریف کلیتر، فرآیندهای همکار به فرآیندهایی اطلاق میشود که بتوانند از دیگر فرآیندهای در حال اجرا تاثیر بگیرند یا روی آنها تاثیر بگذارند.

برخی دلایل برای تامین یک محیط همکاری بین فرآیندها عبارتند از:

۱. **اشتراک گذاری اطلاعات (Information Sharing)**: وجود انگیزه در چندین برنامه کاربردی برای دسترسی همروند به یک قطعه اطلاعاتی مشترک،
۲. **تسریع محاسبات (Computation Speedup)**: اگر یک تسک را بخواهیم سریعتر اجرا کنیم میبایست آن را به زیر تسکهایی شکسته و هرکدام را به طور موازی با یکدیگر اجرا کنیم. برای یکپارچگی و عملکرد صحیح سیستم در چنین شرایطی نیازمند محیط کاری مشترک برای کنترل و مدیریت زیر تسکها و در نهایت تسک اصلی هستیم.
۳. **ماژولار یا پیمانهای بودن (Modularity)** : ممکن است در سیستمی تمایل داشته باشیم که توابع سیستم را در چندین دسته از فرآیندها یا نخهای مجزا تقسیمبندی کنیم در چنین حالتی برای داشتن عملکرد صحیح و یکپارچه داشتن محیط همکاری مشترک ضروری است.

فرآیندهای Cooperate به منظور تبادل داده به (ارسال/دریافت داده به/از فرآیندهای دیگر نیاز به ارتباط درون فرآیندی (IPC) دارند. دو مدل پایه برای چنین