Department of
Computer Engineering

Homework 1-Solution

Operating Systems

Fall 2023

Dr. Javadi

**پاسخ سوال ۱:**

الف)

Direct Memory Access یک روش برای انتقال داده‌ها بین دستگاه‌های ورودی/خروجی (I/O) و RAM می‌پردازد. در سیستم‌های کامپیوتری، وقتی یک دستگاه I/O مانند کارت گرافیک، کارت شبکه یا دستگاه ذخیره‌سازی داده‌هایی را برای انتقال به حافظه مرکزی (RAM) دارد، ممکن است نیاز باشد که از طریق پردازنده عملیات انتقال را انجام دهد. این کار می‌تواند باعث لود بالای پردازنده شود و باعث کاهش عملکرد سیستم شود.

به کمک DMA، این مشکل را می‌توان به حداقل رساند. DMA یک کنترلر جانبی است که به طور مستقیم به I/O و حافظه می‌تواند دسترسی داشته باشد و بدون نیاز به مداخله پردازنده اصلی، داده‌ها را بین آنها منتقل کند.

ب)

برای طراحی یک سیستم که به کاربر امکان انتخاب یک سیستم عامل از بین چند گزینه را در هنگام بوت شدن می‌دهد، شما به یک نرم‌افزار مدیریت بوت (Boot Manager) نیاز دارید. این نرم‌افزار باید در مرحله‌ی بوت‌شدن از کامپیوتر اجرا شود و به کاربر اجازه دهد تا سیستم عامل مورد نظر خود را انتخاب کند.

تئوری اصلی اینجا این است که هنگامی که کامپیوتر را روشن می‌کنید، نرم‌افزار مدیریت بوت اجازه می‌دهد که سیستم عامل مورد نظر را انتخاب کنید. برنامه‌ی bootstrap نقش اصلی در اینجا را ایفا نمی‌کند؛ به جای آن، برنامه‌های مدیریت بوت مانند GRUB برای انتخاب و بوت کردن سیستم عامل مسئول هستند.

ج)

حالت کرنل (Kernel Mode) و حالت کاربر (User Mode) دو حالت اصلی در سیستم‌عامل هستند که به حفاظت و امنیت سیستم کمک می‌کنند.

- حالت کرنل دسترسی کلی به سخت‌افزار و منابع سیستم دارد و مسئول مدیریت منابع و اجرای وظایف مهمی مانند مدیریت حافظه فیزیکی، درایورهای سخت‌افزاری و سیاست‌های امنیتی است.

- حالت کاربر محدودتر است و دسترسی به منابع حساس و سخت‌افزاری سیستم را ندارد. بیشتر وظایف برنامه‌های کاربردی در این حالت اجرا می‌شوند. بنابراین وجود دو حالت باعث می شود که در تمام مواقع ما در حالت کرنل نباشیم که بتوانیم به منابع حساس و عملیات های حساس دسترسی داشته باشیم.

- a. Set value of timer:

حالت کرنل - تنظیم تایمر سیستم مستلزم دسترسی به سخت‌افزار است.

- b. Read the clock:

حالت کاربر - خواندن ساعت سیستم یک عملیات غیر حساس است و می‌تواند در حالت کاربر انجام شود.

- c. Clear memory:

حالت کرنل - پاک کردن حافظه فیزیکی نیاز به دسترسی به منابع سخت‌افزاری دارد.

- d. Issue a trap instruction:

حالت کرنل - دستورهای تله معمولاً برای اجازه دادن به برنامه‌های کاربری برای اجرای وظایف سیستمی در حالت کرنل استفاده می‌شوند.

- e. Turn off interrupts:

حالت کرنل - کنترل تعطیل‌کردن حالت‌های وقفه به عهده حالت کرنل است.

- f. Modify entries in device-status table:

حالت کرنل - تغییرات در جداول وضعیت دستگاه برای ارتباط با سخت‌افزار مشخص نیاز به دسترسی حالت کرنل دارد.

- g. Switch from user to kernel mode:

حالت کرنل - تغییر حالت اجرایی از حالت کاربر به حالت کرنل نیازمند دسترسی حالت کرنل است.

- h. Access I/O device:

حالت کرنل - دسترسی به دستگاه‌های ورودی/خروجی معمولاً توسط حالت کرنل کنترل می‌شود.

د)

در یک محیط multiprogramming و sharing-time که چندین کاربر به صورت همزمان سیستم را به اشتراک می‌گذارند، ممکن است مشکلات امنیتی مختلفی به وجود آید. دو مورد از این مشکلات عبارتند از:

۱. دسترسی غیرمجاز: به منابع وقتی که چندین کاربر به صورت همزمان از سیستم استفاده می‌کنند، این امر ممکن است منجر به دسترسی غیرمجاز به منابع شود. برای مثال، یک کاربر می‌تواند تلاش کند تا به منابعی دسترسی پیدا کند که مخصوص به دیگر کاربران است، یا حتی به محدودیت‌ها و مجوزهای سیستم تجاوز کند. این مشکل می‌تواند اطلاعات حساس را در معرض خطر قرار دهد.

۲. تداخل در اشتراک منابع: وقتی چند کاربر به صورت همزمان از منابع سیستم استفاده می‌کنند، ممکن است تداخل‌ها و اشتباهات به وجود بیاید. مثلاً اگر یک کاربر به صورت همزمان تلاش کند تا یک فایل را ویرایش کند و کاربر دیگر همزمان تغییرات را ذخیره کند، این ممکن است منجر به از دست رفتن اطلاعات یا خرابی فایل شود.

۳. انتقال اطلاعات نادرست: وقتی چندین کاربر از سیستم به صورت همزمان استفاده می‌کنند، ممکن است اطلاعات حساس یا مهم به طور نادرست از یک فرآیند به فرآیند دیگر منتقل شود. این انتقالات نادرست می‌توانند باعث افشای اطلاعات حساس و تخلف در حریم شخصی کاربران شوند.

۴. کنترل نادرست: کاربران ممکن است تلاش کنند تا کنترل سیستم را به دست بگیرند و تغییرات نادرستی در تنظیمات یا تنظیمات امنیتی سیستم اعمال کنند. این می‌تواند به خطر امنیتی سیستم و تخریب آن منتهی شود.

**پاسخ سوال ۲:**

الف ) وقفه به معنای توقف یا انقطاع در یک فرآیند یا سیستم است.

وقفه‌ها را می‌توان به دو دسته اصلی تقسیم کرد: وقفه‌های سنکرون و وقفه‌های آسنکرون.

۱. وقفه‌های سنکرون: وقفه‌های سنکرون در یک زمان معین یا زمانی خاص رخ می‌دهند. به عبارت دیگر، زمان وقوع وقفه‌های سنکرون قبل‌از-زمانی خاص یا بعد از-زمانی خاص مشخص می‌شود. این وقفه‌ها معمولاً با سیگنال‌های ساعتی یا رخدادهای تایمینگ شده هماهنگ می‌شوند.

۲. وقفه‌های آسنکرون:

وقفه‌های آسنکرون به صورت غیرمنظم و بدون یک زمان معین و بدون تنظیم از پیش اتفاق می‌افتند. این وقفه‌ها ناشی از وقایع خارجی یا درخواست‌های کاربر هستند.

مقایسه وقفه‌های سنکرون و آسنکرون:

۱. زمان وقوع:

- وقفه‌های سنکرون: زمان وقوع این وقفه‌ها قبل‌از-زمانی خاص یا بعد از-زمانی خاص مشخص می‌شود.
- وقفه‌های آسنکرون: زمان وقوع این وقفه‌ها غیرقابل پیش‌بینی و غیرمنظم است.

۲. هماهنگی:

- وقفه‌های سنکرون معمولاً با سیگنال‌های ساعتی یا رخدادهای تایمینگ شده هماهنگ می‌شوند.

- وقفه‌های آسنکرون به صورت مستقل از سیگنال‌های هماهنگی رخ می‌دهند و به وقایع خارجی وابسته‌اند.

۳. کاربرد:

- وقفه‌های سنکرون معمولاً در سیستم‌های با زمان‌بندی دقیق مانند میکروکنترلرها و RTOS استفاده می‌شوند.

- وقفه‌های آسنکرون به عنوان وقفه‌های عمومی در سیستم‌های کامپیوتری و برنامه‌های نرم‌افزاری عمومی مورد استفاده قرار می‌گیرند.

ب) Interrupt و Trap هر دو به عنوان مکانیسم‌های مهم برای مدیریت وقفه‌ها در سیستم‌های کامپیوتری استفاده می‌شوند، اما تفاوت‌های مهمی دارند. در ادامه، تفاوت‌های اصلی بین Interrupt و Trap را توضیح می‌دهیم:

Interrupt:

۱. منبع:

- Interrupt اغلب از منابع خارجی به سیستم کامپیوتری وارد می‌شود، مثل وقوع یک وقفه سخت‌افزاری نظیر نوسان برق یا از درخواست‌های دستگاه‌های جانبی مانند کیبورد یا ماوس.

۲. زمان وقوع:

- Interruptها به طور ناگهانی و ناپیش‌بینی رخ می‌دهند. زمان وقوع این وقفه‌ها توسط منبع ایجاد کننده آنها تعیین می‌شود.

۳. عملکرد:

- Interruptها به سیستم عامل اجازه می‌دهند تا از وقوع وقفه با خبر شود و وظایف مربوط به آن را انجام دهد. این وظایف می‌تواند تغییر وضعیت برنامه، خواندن و نوشتن داده‌ها به یک دستگاه جانبی یا انجام عملیات‌های مهم دیگر باشد.

:Trap

۱. منبع:

- Trapها اغلب از داخل برنامه‌ها یا نرم‌افزار ایجاد می‌شوند. به عبارت دیگر، این وقفه‌ها از داخل نرم‌افزار ایجاد و کنترل می‌شوند.

۲. زمان وقوع:

- Trapها به صورت پیش‌بینی شده در زمان اجرای برنامه رخ می‌دهند. این وقفه‌ها در نقاط خاصی از برنامه توسط دستورهای مشخص ایجاد می‌شوند.

۳. عملکرد:

- Trapها به برنامه در حال اجرا اجازه می‌دهند تا وظایف خاصی را انجام دهند. این می‌تواند شامل تغییر وضعیت برنامه، فراخوانی توابع یا خدمات سیستمی، اعلان خطاها یا انجام عملیات‌های مشخص در محیط نرم‌افزاری باشد. در کل، Interruptها اغلب به عنوان وقفه‌های سخت‌افزاری وارد می‌شوند و توسط سیستم عامل به عنوان وقفه‌های خارجی مدیریت می‌شوند. Trapها بیشتر به عنوان وقفه‌های نرم‌افزاری در نقاط خاص از برنامه تولید می‌شوند و معمولاً به کنترل دقیق برنامه و اجرای آن کمک می‌کنند.

پ) فرآیند مدیریت یک وقفه از لحظه ایجاد تا اتمام آن مرحله به مرحله توضیح داده شده است:

۱. ایجاد وقفه:

- وقفه توسط منبع خارجی (مثلاً یک دستگاه جانبی) یا نرم‌افزار (مثلاً اجرای یک دستور Trap) ایجاد می‌شود.

۲. اطلاع‌رسانی به سیستم:

- در این مرحله، سیستم متوجه وقوع وقفه می‌شود. این اطلاع‌رسانی به سیستم عامل واگذار می‌شود. اگر وقفه از نرم‌افزاری ایجاد شده باشد (Trap)، سیستم عامل مسئول اجرای دستور Trap است.

۳. ذخیره وضعیت فعلی:

- برای ادامه اجرای برنامه اصلی بعد از اتمام وقفه، وضعیت فعلی (مانند محتوای ثبت‌های مهم) بر روی استک (Stack) ذخیره می‌شود تا در آینده بتوان به وضعیت فعلی بازگشت داد.

۴. اجرای رویداد وقفه:

- در این مرحله، وظایف مربوط به وقفه اجرا می‌شوند. اگر وقفه مربوط به دستگاه جانبی باشد، دستگاه به سیستم عامل اطلاع می‌دهد که درخواست انجام شده است و سیستم عامل وظایف مربوط به دستگاه را انجام می‌دهد. اگر وقفه از طریق دستور Trap ایجاد شده باشد، دستور مربوط به Trap اجرا می‌شود.

۵. پایان وقفه:

- پس از اتمام وظایف مربوط به وقفه، وضعیت فعلی به وضعیت قبلی بازگردانده می‌شود. این به معنای بازگشت به اجرای برنامه اصلی است.

۶. ادامه اجرای برنامه:

- سیستم به اجرای برنامه کاربری ادامه می‌دهد و از وضعیتی که قبل از وقوع وقفه داشته بهره می‌برد.

توجه داشته باشید که در صورتی که CPU مشغول انجام برنامه کاربری باشد و وقفه ایجاد شود، CPU متوقف می‌شود و باید وظایف مربوط به وقفه انجام شود. سیستم عامل و سخت‌افزار معمولاً از مکانیزم‌هایی مانند استک برای مدیریت و بازگشت به وضعیت قبلی استفاده می‌کنند تا از از دست رفتن اطلاعات مهم جلوگیری کنند و اجازه دهند که وظایف وقفه به درستی انجام شوند.

**پاسخ سوال ۳:**

الف) Running -> ready

ب) running -> waiting

پ) waiting -> ready

ت) running -> terminated

ث) waiting -> ready

**پاسخ سوال ۴:**

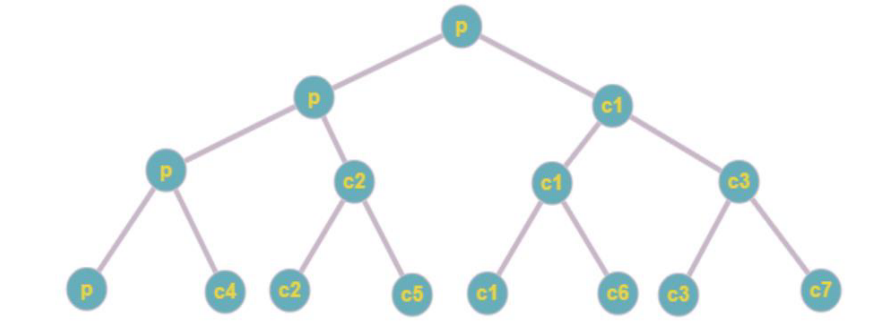
الف)

در ابتدا، پردازه اصلی (پردازه والد) شروع به اجرای حلقه می‌کند.

در اولین مرور حلقه ، یک فرزند ایجاد می‌شود و حلقه از ابتدا شروع می‌شود. حالا دو پردازه والد و پردازه فرزند وجود دارند.

در مرحله دوم حلقه ، هر کدام از پردازه والد و پردازه فرزند دوباره تابع fork() را فراخوانی می‌کنند و بنابراین تعداد پردازه‌ها تا چهار تا افزایش می‌یابد. این باعث ایجاد سه پردازه فرزند جدید می‌شود و حالا داریم یک پردازه والد و چهار پردازه فرزند.

ادامه اجرای حلقه توسط همه پردازه‌ها، تعداد دیگری از پردازه‌ها را ایجاد می‌کند. در نهایت، تعداد کل پردازه‌ها به ۸ پردازه می‌رسد. در نتیجه ۸ بار عبارت Hello چاپ میشود.

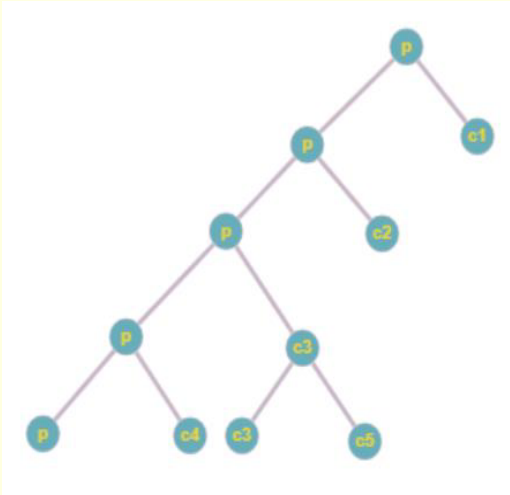



ب)

ابتدا یک پردازه والد و یک پردازه فرزند ایجاد می‌شوند.

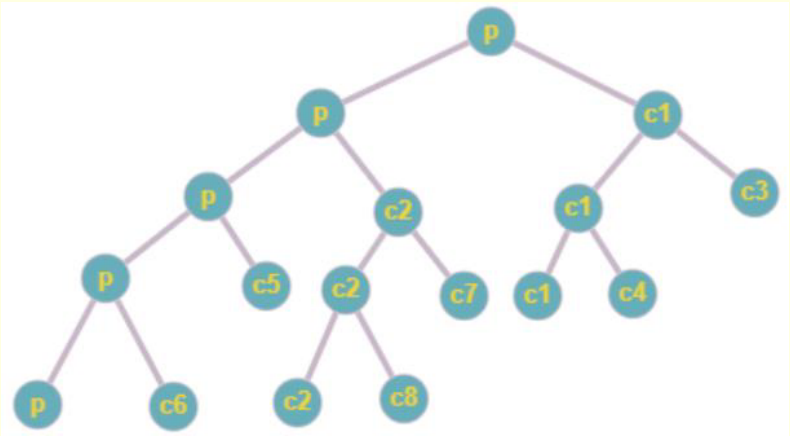
پردازه فرزند و والد هردو به بخش داخل if وارد نمی‌شود زیرا برای وارد شدن والد به داخل if باید فورک دوم هم عددی بزرگتر از صفر برگرداند و پردازه فرزند هم در حال حاضر صفر برگردانده پس هردو وارد if نمیشوند ، تا به اینجا یکبار A چاپ میشود.

حال فورک بعدی اتفاق میفتد و باز هم پردازه فرزند وارد if نشده و عبور میکند اما این دفعه پردازه والد میتواند وارد if بشود. در داخل if دو بار دیگر عمل فورک انجام میشود که باعث به وجود آمدن سه پردازه دیگر هم میشود پس بنابر این در آخر به طور کلی ۶ عدد پردازه بوجود میاید و ۶ بار عبارت A چاپ میشود.




ج)

در اولین دور اجرای حلقه وقتی پردازه به شرط if میرسد، یک فورک انجام میدهد که پردازه فرزند به داخل if وارد نمیشود اما والد به داخل if میرود و یکبار دیگر عمل فورک را انجام میدهد و سپس دو پردازه دیگر ساخته میشود که هردو B را چاپ میکنند، در حال حاضر سه پردازه برای بار دوم حلقه را اجرا میکنند و طبق همین مراحل در آخر به طور مجموع ۹ پردازه ایجاد شده و عبارت داخل if ۸ بار چاپ میشود.

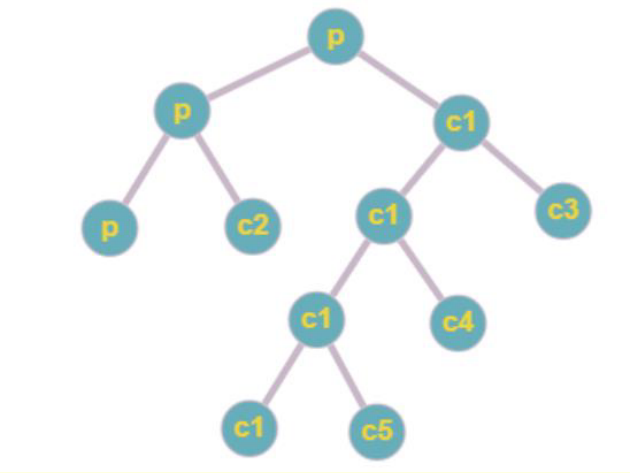


د)

در این گونه سوالات که عملیات منطقی با fork ترکیب شده اند کافیست یک پرانتز فرضی برای عباراتی که عملیات && دارند در نظر بگیریم و سپس سوال را حل کنیم یعنی به این صورت عبارت داخل if را در نظر بگیریم:

```
fork() || ( fork() && fork() )
```

در اینجا ابتدا پردازه اصلی یکبار فورک میشود و پردازه والد به داخل if میرود و یکبار دیگر عملیات فورک را انجام میشود و پردازه تشکیل شده و والد ، عبارت داده شده را چاپ میکنند. اما پردازه فرزند حاصل شده توسط اولین فورک، به داخل if نمیرود و فورک دوم را اجرا میکند که در این صورت اگر والد نبود در کل وارد نمیشود اما اگر بود فورک سوم هم انجام میشود، درصورتی که عددی بزرگتر از صفر برگردانده شود به داخل if رفته و یکبار دیگر فورک انجام میدهد و پردازه ایجاد شده و والد آن، عبارت داده شده را چاپ میکنند. در آخر مجموعا ۶ پردازه داریم و ۴ بار C چاپ میشود.



**پاسخ سوال ۵:**

الف)

مقدار چاپ شده صفر میباشد زیرا پردازه فرزند pcb جداگانه ای از پردازه والد دارد و مقدار کپی شده مخصوص خود را از متغیر count در خود دارد.

ب)

۱:

**Zombie:** پردازه ای که خاتمه یافته است، اما والد آن هنوز()wait را فراخوانی نکرده است به عنوان یک پردازه زامبی شناخته می شود.

**Orphan:** پردازه ای که والد آن دیگر وجود ندارد، یعنی بدون فراخوانی ()wait برای پایان پردازه فرزند، به پایان رسیده یا خاتمه یافته است، فرآیند یتیم یا orphan نامیده می شود.

۲:

- قطعه کد سمت راست :

والد اجرا را به پایان می رساند و در حالی که پردازش فرزند هنوز در حال اجرا است خارج می شود این به این معنی میباشد که در این قطعه کد، پردازه Orphan ایجاد میشود.

- قطعه کد سمت چپ:

فرزند اجرای خود را با استفاده از فراخوانی سیستمی ()exit به پایان می‌رساند در حالی که والد به مدت ۶۰ ثانیه در حالت sleep قراردارد بنابراین ()wait را فراخوانی نمی‌کند و ورودی پردازش فرزند همچنان در جدول پردازش وجود دارد ، با توجه به این موضوع میتوان گفت که این قطعه کد پردازه Zombie ایجاد میکند.