«گزارش کار پروژه سوم آزمایشگاه »

«Operating System»

البرز محمودیان 810101514

کسری کاشانی نژاد 810101490

نرگس بابالار 810101557

سوال 1:

ساختار PCB و همچنین وضعیت‌های تعریف‌شده برای هر پردازه را در XV6 پیدا کرده و گزارش کنید. آیا شباهتی میان ساختار و ساختار به تصویر کشیده‌شده در شکل ۳.۳ منبع درس وجود دارد؟ (ذکر حداقل دو مورد و معادل آن‌ها درXV6)

پاسخ:

در سیستم‌عاملXV6 ، PCB (Process Control Block)نماینده یک پردازه است که اطلاعات مختلفی درباره پردازه‌ها مانند شناسه پردازه(PID) ، اشاره‌گرها به فضای آدرس پردازه، وضعیت اجرای پردازه و زمان‌بندی پردازه در آن ذخیره می‌شود. درXV6 ، ساختار PCB به صورت struct procدر فایل proc.hتعریف شده است.

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int used_syscalls[MAX_SYSCALLS];
  struct timeInfo ti;
};
```

شباهت‌ها بین ساختار procدر XV6 و شکل ۳.۳:

1. **وضعیت پردازه :(State)** در شکل ۳.۳، پردازه‌ها وضعیت‌هایی نظیر Waiting، Readyو Runningدارند. معادل این وضعیت‌ها در ساختار procدر XV6 فیلدی به نام stateاست که این حالات را با مقادیر زیر نشان می‌دهد:

- RUNNING
- SLEEPING
- RUNNABLE

2. **شناسه پردازه :(PID)** در هر دو سیستم، هر پردازه یک شناسه یکتا دارد. در XV6 این شناسه در فیلدی به نام pidذخیره می‌شود.

سوال 2:

هر کدام از وضعیت‌های تعریف معادل کدام وضعیت در شکل ۱ می‌باشند؟

پاسخ:

در سیستم‌عاملXV6 ، وضعیت‌های پردازه در فیلد stateاز ساختار procذخیره می‌شوند. این وضعیت‌ها و معادل آن‌ها در شکل ۱ به شرح زیر هستند:

1. UNUSEDدر:(XV6)
   - این وضعیت مربوط به پردازه‌هایی است که تخصیص داده نشده‌اند یا از چرخه پردازه‌ها خارج شده‌اند.
   - معادل در شکل ۱ :این وضعیت معادل خاصی در چرخه شکل ۱ ندارد زیرا پردازه هنوز به چرخه وارد نشده است.
2. EMBRYOدر:(XV6)
   - پردازه‌ای که تازه ایجاد شده و هنوز آماده اجرا نشده است.
   - معادل در شکل ۱ :این وضعیت معادل وضعیت Newدر شکل ۱ است.
3. SLEEPINGدر:(XV6)
   - پردازه در حال انتظار برای یک رویداد I/O یا سایر شرایط است.
   - معادل در شکل ۱ :این وضعیت معادل Waitingدر شکل ۱ است.
4. RUNNABLEدر:(XV6)
   - پردازه آماده اجرا بوده و منتظر تخصیص CPU است.
   - معادل در شکل ۱ :این وضعیت معادل Readyدر شکل ۱ است.

5. RUNNING (در XV6):
   - پردازه در حال اجرا روی CPU است.
   - معادل در شکل ۱ :این وضعیت معادل Runningدر شکل ۱ است.
6. ZOMBIE (در XV6):
   - پردازه‌ای که اجرا را تمام کرده اما هنوز منابع آن آزاد نشده است.
   - معادل در شکل ۱ :این وضعیت معادل Terminatedدر شکل ۱ است.

در XV6 برخی وضعیت‌ها مانند UNUSEDبرای مدیریت منابع سیستم‌عامل اضافه شده‌اند و در چرخه استاندارد پردازه‌ها در شکل ۱ نمایش داده نشده‌اند.

سوال 3:

با توجه به توضیحات گفته‌شده، کدام یک از توابع موجود در proc.c منجر به انجام گذار از حالت new به حالت ready که در شکل ۱ به تصویر کشیده شده خواهد شد؟ وضعیت یک پردازه در XV6 در این گذار از چه حالتی/حالت‌هایی به چه حالتی/حالت‌هایی تغییر می‌کند؟ پاسخ خود را با شکل ۱ مقایسه کنید.

پاسخ:

گذار از حالت newبه readyدرXV6 :

در سیستم‌عاملXV6 ، گذار از وضعیت newبه readyتوسط تابعی انجام می‌شود که پردازه جدید ایجاد و مقداردهی اولیه شده و سپس برای اجرا آماده شود. این تابع allocprocدر فایل proc.cاست. این تابع مسئول تخصیص یک پردازه جدید و مقداردهی اولیه به آن است.

- جزئیات عملکرد allocproc:

1. ابتدا یک پردازه در وضعیت UNUSEDجستجو شده و به وضعیت EMBRYOمنتقل می‌شود.
2. مقادیر اولیه موردنیاز (مانند PID و اشاره‌گر به فضای آدرس)در پردازه تنظیم می‌شوند.
3. پس از مقداردهی، پردازه به وضعیت RUNNABLEمنتقل می‌شود، که معادل وضعیت readyدر شکل ۱ است.

- مقایسه با شکل ۱:
  - در شکل ۱، گذار از وضعیت newبه readyمعادل ایجاد یک پردازه جدید و آماده‌سازی آن برای اجرا است.
  - درXV6 ، این گذار با تغییر وضعیت از UNUSEDبه RUNNABLEاز طریق (EMBRYO)انجام می‌شود.

تابع allocprocمسئول این گذار است. مراحل آن به طور کامل با روند نمایش داده‌شده در شکل ۱ تطابق دارد، با این تفاوت که در XV6 وضعیت میانی **EMBRYO**نیز وجود دارد، که در شکل ۱ به طور مستقیم نمایش داده نشده است.

سوال 4:

سقف تعداد پردازه‌های ممکن در XV6 چه عددی است؟ در صورتی که یک پردازه تعداد زیادی پردازه فرزند ایجاد کند و از این سقف عبور کند، کرنل چه واکنشی نشان داده و برنامه سطح کاربر چه بازخوردی دریافت می‌کند؟

پاسخ:

سقف تعداد پردازه‌ها درXV6 :

در سیستم‌عاملXV6 ، تعداد حداکثری پردازه‌ها توسط ماکروی NPROCدر فایل param.h تعریف شده است. مقدار پیش‌فرض این ماکرو **64** است، به این معنا که در هر لحظه حداکثر **64** پردازه می‌توانند در سیستم وجود داشته باشند (شامل پردازه‌های کاربر و کرنل).

```
#define NPROC        64  // maximum number of processes
#define KSTACKSIZE 4096  // size of per-process kernel stack
#define NCPU          8  // maximum number of CPUs
#define NOFILE       16  // open files per process
#define NFILE       100  // open files per system
#define NINODE       50  // maximum number of active i-nodes
#define NDEV         10  // maximum major device number
#define ROOTDEV       1  // device number of file system root disk
#define MAXARG       32  // max exec arguments
#define MAXOPBLOCKS  10  // max # of blocks any FS op writes
#define LOGSIZE      (MAXOPBLOCKS*3)  // max data blocks in on-disk log
#define NBUF         (MAXOPBLOCKS*3)  // size of disk block cache
#define FSSIZE       1000  // size of file system in blocks
#define SYS_TICK 10
#define QUANTUM 50
#define DEFAULT_BURST_TIME 2
#define DEFAULT_CONFIDENCE 50
#define STARVATION_BOUNDRY 800
#define TIME_SLICE_UNIT 100
#define RR_WEIGHT 3
#define SJF_WEIGHT 2
#define FCFS_WEIGHT 1
```

- **واکنش کرنل در صورت عبور از سقف:**

1. زمانی که یک پردازه (فرزند) جدید ایجاد شود، کرنل ابتدا فضای خالی در جدول پردازه‌ها (ptable) را جستجو می‌کند. اگر تمام ورودی‌های جدول پردازه‌ها پر باشند (یعنی سقف NPROC پر شده باشد) :
   - تابعی که مسئول تخصیص پردازه جدید است مانند (allocproc)شکست خورده و مقدار NULLبازمی‌گرداند.
   - پردازه فرزند ایجاد نمی‌شود و منابعی برای آن تخصیص داده نخواهد شد.
2. این موضوع در کرنل ممکن است به صورت خطای داخلی ثبت شود، اما هیچ کرش یا اختلالی در عملکرد کرنل ایجاد نمی‌کند.

- **واکنش برنامه سطح کاربر:**

برنامه سطح کاربر که درخواست ایجاد پردازه جدید را داده است (معمولاً از طریق فراخوانی سیستمی fork) :

- مقدار بازگشتی forkرا بررسی می‌کند. اگر مقدار بازگشتی 1-باشد، این به معنای شکست در ایجاد پردازه جدید است.
- برنامه می‌تواند این خطا را مدیریت کرده و پیام خطایی به کاربر جهت محدودیت ایجتد پردازنده جدید نمایش دهد.

سوال 5:

چرا نیاز است در ابتدای هر حلقه تابع scheduler، جدول پردازه‌ها قفل شود؟ آیا در سیستم‌های تک‌پردازنده‌ای هم نیاز است این کار صورت بگیرد؟

پاسخ:

دلیل قفل کردن جدول پردازه‌ها در ابتدای هر حلقه scheduler:

1. **حفظ هماهنگی بین پردازه‌ها:**

- جدول پردازه‌ها (ptable) یک منبع اشتراکی است که توسط بخش‌های مختلف سیستم‌عامل مانند تابع scheduler و توابعی مثل allocproc، fork یا exit دسترسی پیدا می‌کند. اگر چندین پردازه یا عملیات کرنلی به صورت همزمان به جدول پردازه‌ها دسترسی پیدا کنند، ممکن است باعث تداخل در داده‌ها، خرابی اطلاعات یا رفتارهای پیش‌بینی‌نشده شود.
- قفل کردن جدول پردازه‌ها از چنین شرایطی جلوگیری می‌کند و تضمین می‌کند که در هر لحظه فقط یک بخش از کد به این منبع دسترسی داشته باشد.

2. تضمین امنیت تغییرات:

- زمانی که scheduler به جدول پردازه‌ها دسترسی پیدا کرده و وضعیت پردازه‌ای را تغییر می‌دهد یا پردازه‌ای را انتخاب می‌کند، نیاز است که این عملیات بدون تداخل انجام شود. قفل جدول پردازه‌ها مانع تغییرات همزمان توسط عملیات‌های دیگر می‌شود.

آیا در سیستم‌های تک‌پردازنده‌ای نیز این قفل لازم است؟

بله، حتی در سیستم‌های تک‌پردازنده‌ای نیز نیاز به قفل کردن جدول پردازه‌ها وجود دارد. دلایل آن عبارتند از:

1. رویدادهای وقفه:

- در سیستم‌های تک‌پردازنده‌ای، وقفه‌ها می‌توانند در هر زمانی رخ دهند و ممکن است در طی اجرای کدی که به جدول پردازه‌ها دسترسی دارد، کنترل به یک عملیات دیگر داده شود (مانند یک وقفه I/O یا وقفه زمان‌سنج) این عملیات ممکن است به جدول پردازه‌ها دسترسی پیدا کرده و باعث تداخل شود. قفل کردن جدول پردازه‌ها از چنین شرایطی جلوگیری می‌کند.

2. **پشتیبانی از چندوظیفگی**:

- اگرچه سیستم تک‌پردازنده‌ای به طور همزمان فقط یک پردازه را اجرا می‌کند، اما همچنان از چندوظیفگی پشتیبانی می‌کند. در این حالت، ممکن است کرنل در حین تغییر زمینه (context switch) نیاز به دسترسی به جدول پردازه‌ها داشته باشد. قفل کردن جدول تضمین می‌کند که وضعیت پردازه‌ها در طی این فرآیند ثابت باقی می‌ماند.

3. قابلیت توسعه به سیستم چندپردازنده‌ای:

- استفاده از قفل حتی در سیستم تک‌پردازنده‌ای، طراحی سیستم‌عامل را برای انتقال به سیستم‌های چندپردازنده‌ای آسان‌تر می‌کند، زیرا مکانیزم‌های مدیریت منابع و همگام‌سازی از قبل در نظر گرفته شده‌اند.

سوال6:

با فرض اینکهxv6 در حالت تک هسته ای در حال اجراست اگر یک پردازنده به حالت RUNNABLE برود و صف پردازنده در حال طی شدن باشد (proc::335) در مکانیزم زمان بندیxv6 نسبت به موقعیت پردازنده در صف،در چهiteration ای امکانschedule پیدا میکند ؟(در همان iteration یا دریteration بعدی)

پاسخ:

پردازه‌ای که وضعیتش از SLEEPINGیا هر وضعیت دیگر به RUNNABLEتغییر می‌کند و در حین زمان‌بندی قرار دارد، در iterationبعدی ممکن است توسط زمان‌بند انتخاب شود، زیرا در همان iteration فعلی پردازه‌هایی که قبلاً در صف قرار داشته‌اند، مورد ارزیابی و انتخاب قرار خواهند گرفت.

- در XV6، مکانیزم زمان‌بندی مبتنی بر first-come, first-served (FCFS)وround-robin است. زمانی که پردازه‌ای وضعیتش از SLEEPINGیا هر وضعیت دیگری به RUNNABLE تغییر پیدا می‌کند، در صورتی که پردازه‌ها در حال ارزیابی (iterate) در صف باشند، در همان iterationممکن است انتخاب نشود.
- این پردازه به صف readyاضافه می‌شود، ولی بسته به موقعیت آن در صف، در iteration بعدی که زمان‌بند پردازه‌ها را دوباره بررسی می‌کند، ممکن است انتخاب شود.
- زمان‌بند در حال ارزیابی و انتخاب پردازه‌ها است و پردازه‌های موجود در صف (که در حال حاضر RUNNABLEهستند) در همان iteration بررسی می‌شوند، پردازه‌ای که به حالت RUNNABLEتغییر وضعیت داده است، وقتی به صف اضافه می‌شود، در iterationبعدی مورد ارزیابی قرار می‌گیرد.

سوال 7:

رجیسترهای موجود در ساختار contextرا نام ببرید.

پاسخ:

در سیستم‌عامل XV6، ساختار contextشامل اطلاعاتی از رجیسترها است که برای انجام تعویض متن (context switch)ذخیره و بازیابی می‌شوند. این رجیسترها شامل موارد زیر هستند:

1. edi
2. esi
3. ebx
4. ebp
5. eip

```
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

سوال 8:

همانطور که می‌دانید یکی از مهم‌ترین رجیسترها قبل از هر تعویض متن Program Counterاست که نشان می‌دهد اجرای برنامه تا کجا پیش رفته است. با ذخیره‌سازی این رجیستر چگونه از ادامه برنامه را بازیابی کرد؟ این رجیستر در ساختار contextچه نقشی دارد؟ آیا در انجام تعویض متن ذخیره می‌شود؟

پاسخ:

چگونگی ذخیره و بازیابی Program Counter:

1. هنگام تعویض متن:(Context Switch)

- وضعیت پردازه جاری، شامل تمام رجیسترهای آن از جمله (Program Counter)، در ساختار contextذخیره می‌شود.
- این عمل توسط تابعی مانند swichانجام می‌شود که وضعیت رجیسترها را از پردازه فعلی ذخیره کرده و وضعیت رجیسترها را برای پردازه جدید بازیابی می‌کند.

2. هنگام بازگشت به پردازه:
   - مقدار ذخیره‌شده در Program Counterاز ساختار contextبازیابی می‌شود.
   - این مقدار به CPU ارسال می‌شود تا اجرای پردازه از نقطه ذخیره‌شده ادامه یابد.

نقش Program Counterدر ساختار context:

- در XV6، رجیستر EIP (Instruction Pointer)به‌عنوان نماینده Program Counterعمل می‌کند و در ساختار contextذخیره می‌شود.
- این رجیستر تضمین می‌کند که هنگام بازگشت به پردازه، اجرای آن دقیقاً از آخرین دستورالعملی که اجرا شده بود، ادامه پیدا کند.

سوال 9:

همانطور که در قسمت قبل مشاهده کردید، ابتدای تابع scheduler، ایجاد وقفه به کمک تابع stiفعال می‌شود. با توجه به توضیحات این قسمت، اگر وقفه‌ها فعال نمی‌شد، چه مشکلی به وجود می‌آمد؟

پاسخ:

اگر وقفه‌ها در تابع schedulerفعال نشوند:

1. وقفه تایمر اجرا نمی‌شود، زمان‌بندی پردازه‌ها مختل شده و پردازه‌ها به ترتیب عادلانه اجرا نمی‌شوند.
2. سیستم نمی‌تواند به وقفه‌های I/O و سخت‌افزاری پاسخ دهد، که ممکن است منجر به قفل شدن شود.
3. تغییر پردازه (context switch) انجام نمی‌شود و یک پردازه ممکن است CPU را انحصاری اشغال کند.

در نتیجه، سیستم دچار اختلال یا توقف می‌شود.

سوال 10:

به نظر شما وقفه تایمر هر چند مدت یک بار صادر می‌شود؟ (راهنمایی: می‌توانید با اضافه کردن یک cprintf پس از ++ticks این موضوع را مشاهده کنید.)

پاسخ:

وقفه تایمر در XV6 تقریباً هر 10 میلی‌ثانیه یک بار اجرا می‌شود و این مقدار توسط تنظیمات سخت‌افزاری و سیستم مشخص می‌شود.

سوال 11:

با توجه به توضیحات داده شده، چه تابعی منجر به انجام شدن گذار interrupt در شکل ۱ خواهد شد؟

پاسخ:

تابع yield:

- **yield** به صورت صریح توسط یک پردازه فراخوانی می‌شود تا به کرنل اطلاع دهد که نیاز به تغییر زمینه (context switch) است.
- این تابع به طور دستی پردازه جاری را به **RUNNABLE** تغییر می‌دهد و پردازنده را آزاد می‌کند.
- سپس زمان‌بند (scheduler) یک پردازه جدید را برای اجرا انتخاب می‌کند.
- این تابع پردازنده را آزاد کرده و امکان پردازش وقفه‌ها یا انتخاب پردازه جدید را فراهم می‌کند.

سوال 12:

با توجه به توضیحات قسمت scheduler dispatch می‌دانیم زمان‌بندی در XV6 به شکل نوبت‌گردشی است. حال با توجه به مشاهدات خود در این قسمت، استدلال کنید کوانتوم زمانی این پیاده‌سازی از زمان‌بندی نوبت‌گردشی چند میلی‌ثانیه است؟

پاسخ:

کوانتوم زمانی درXV6 :

1. درXV6 ، وقفه تایمر (IRQ_TIMER) به صورت پیش‌فرض هر 10میلی‌ثانیه یک بار صادر می‌شود.
2. با هر وقفه تایمر، مقدار ticksافزایش می‌یابد و زمان‌بند تصمیم می‌گیرد که آیا پردازه جاری باید متوقف شود یا ادامه یابد.
3. با توجه به این تنظیمات، کوانتوم زمانی در XV6 برابر با 10میلی‌ثانیه است.

سوال 13:

تابع waitدر نهایت از چه تابعی برای منتظر ماندن برای اتمام کار یک پردازه استفاده می‌کند؟

پاسخ:

تابع waitبرای منتظر ماندن جهت اتمام پردازه فرزند، از تابع sleepاستفاده می‌کند که حالت پردازه والد را به حالت SLEEPINGتغییر میدهد.

1 بررسی پردازه‌های فرزند:

- تابع waitابتدا تمام پردازه‌های فرزند پردازه جاری را بررسی می‌کند. این بررسی تحت قفل جدول پردازه‌ها (ptable) انجام می‌شود تا از تداخل با سایر عملیات‌های کرنل جلوگیری شود.

2 حالت zombie:

- اگر پردازه‌ای با وضعیت ZOMBIEیافت شود:
  - منابع مربوط به آن پردازه (مانند حافظه، فایل‌های باز و غیره) آزاد می‌شوند.
  - پردازه از جدول پردازه‌ها حذف می‌شود.
  - تابع waitمقدار بازگشتی را به والد می‌دهد که معمولاً شناسه پردازه فرزند (PID)است.

3 منتظر ماندن:

- اگر هیچ پردازه فرزندی با وضعیت ZOMBIEیافت نشود:
  - تابع sleepفراخوانی می‌شود.
  - این تابع پردازه والد را به حالت SLEEPINGمنتقل می‌کند تا زمانی که وضعیت پردازه‌های فرزند تغییر کند (مانند پایان یافتن یک فرزند و ورود به حالت ZOMBIE).

4 بیدار شدن از sleep:

- زمانی که یکی از پردازه‌های فرزند به وضعیت ZOMBIEتغییر می‌کند، پردازه والد از حالت SLEEPINGخارج شده و مجدداً بررسی می‌کند که آیا پردازه‌ای برای آزاد کردن منابع وجود دارد یا خیر.

سوال 14:

با توجه به پاسخ سوال قبل، استفاده(های) دیگر این تابع چیست؟ (ذکر یک نمونه)

پاسخ:

موارد استفاده تابع wait:

1 منتظر ماندن برای اتمام پردازه فرزند.

2 آزادسازی منابع پردازه‌های فرزند (حافظه، فایل‌ها).

3 مدیریت پردازه‌های فرزند با وضعیت .ZOMBIE

4 جلوگیری از نشت منابع.(Resource Leak)

5 همگام‌سازی بین پردازه والد و فرزند.

سوال 15:

با این تفسیر، چه تابعی در سطح کرنل منجر به آگاه‌سازی پردازه از رویدادی است که برای آن منتظر بوده است؟

پاسخ:

تابع wakeupدر سطح کرنل پردازه‌ای که در حالت SLEEPINGمنتظر یک رویداد بوده است را بیدار کرده و به حالت RUNNABLEمنتقل می‌کند تا پردازنده بتواند آن را اجرا کند.

سوال 16:

با توجه به پاسخ سوال ۹، این تابع منجر به چه وضعیتی در شکل ۱ خواهد شد؟

پاسخ:

yield1 :

- وظیفه این تابع آزاد کردن پردازنده است. وقتی یک پردازه yieldرا فراخوانی می‌کند، وضعیت آن از RUNNINGبه RUNNABLEتغییر می‌کند. این تغییر وضعیت زمانی رخ می‌دهد که پردازه به طور صریح بخواهد اجرای خود را متوقف کند و اجازه دهد سایر پردازه‌ها اجرا شوند.
- بنابراین، yieldتغییر از RUNNINGبه RUNNABLEرا انجام می‌دهد.

wakeup2 :

- وظیفه این تابع بیدار کردن پردازه‌هایی است که منتظر وقوع یک رویداد بوده‌اند .اگر یک پردازه در وضعیت SLEEPING باشد و رویداد مورد انتظار رخ دهد، تابع wakeup وضعیت آن پردازه را از SLEEPING به RUNNABLE تغییر می‌دهد.
- بنابراین، wakeup مسئول تغییر از SLEEPING به RUNNABLE است.

سوال 17:

آیا تابع دیگری وجود دارد که منجر به انجام این گذار شود؟ نام ببرید.

پاسخ:

علاوه بر wakeup، تابع kill نیز می‌تواند باعث انتقال پردازه از SLEEPING به RUNNABLE شود. این تابع در شرایطی استفاده می‌شود که بخواهید پردازه‌ای که در حالت خواب است را مجبور به بیدار شدن کنید و به سیگنال ارسال‌شده پاسخ دهد.

1 ارسال سیگنال به پردازه:

- **kill**برای ارسال یک سیگنال به پردازه موردنظر استفاده می‌شود. این سیگنال معمولاً برای خاتمه دادن به پردازه به کار می‌رود.

2بیدار کردن پردازه در حالت :SLEEPING

- اگر پردازه‌ای در حالت **SLEEPING** باشد (منتظر یک منبع یا رویداد خاص)، فراخوانی **kill** باعث می‌شود که پردازه از این حالت خارج شود.
- این انتقال باعث می‌شود که پردازه به وضعیت **RUNNABLE** تغییر کند و در صف آماده برای اجرا قرار گیرد.

سوال 18:

در بخش ۳.۳.۲ منبع درس با پردازه‌های Orphan آشنا شدید. رویکرد xv6 در رابطه با این گونه پردازه‌ها چیست؟

پاسخ:

در سیستم‌عامل XV6، پردازه‌های Orphan(یعنی پردازه‌هایی که والد آن‌ها از بین رفته است) به پردازه init متصل می‌شوند.

1. زمان Orphan شدن پردازه:
   - وقتی یک پردازه والد (Parent) خاتمه می‌یابد، فرزندان آن پردازه Orphanمی‌شوند، زیرا دیگر والد فعالی برای مدیریت آن‌ها وجود ندارد.
2. رویکرد:XV6
   - برای جلوگیری از رها شدن پردازه‌ها و نشت منابع، XV6تمام پردازه‌های یتیم (Orphan)را به پردازه init متصل می‌کند.
   - پردازه init به عنوان والد جدید این پردازه‌ها عمل کرده و مسئولیت مدیریت آن‌ها، از جمله جمع‌آوری منابع و خاتمه دادن به آن‌ها در صورت نیاز را بر عهده می‌گیرد.

سوال 19:

مقدار CPUS را مجدداً به عدد ۲ بگردانید. آیا همچنان ترتیب که قبلاً مشاهده می‌کردید برجا است؟ علت این امر چیست؟

پاسخ:

همانطور که انتظار می رفت، ترتیب اجرای فرایند ها یکسان است و تغییری نکرده و صرفا فرایند ها میان 2 core تقسیم شده اند.

سوال 20:

در صورت نیاز به مقداردهی اولیه به فیلدهای اضافه شده در ساختارCPU ، در چه تابعی از XV6 بهتر است این کار انجام گیرد؟ (راهنمایی: به main.c مراجعه کنید)

پاسخ:

استفاده از :mpmain

1. اجرای مستقل برای هر:CPU

- تابع mpmain به صورت جداگانه برای هر پردازنده (CPU) اجرا می‌شود. این استقلال به شما امکان می‌دهد فیلدهای مربوط به هر CPU را به طور خاص مقداردهی کنید.

2. مقداردهی پس از آماده‌سازی سیستم:

- برخلاف mainکه تنظیمات اولیه کلی سیستم را انجام می‌دهد، mpmainبعد از اینکه سیستم آماده اجرای وظایف پردازنده‌ها شد، فراخوانی می‌شود. این زمان مناسبی برای مقداردهی دقیق فیلدهای ساختار CPU است.

3. زمان‌بندی مناسب:

- در mpmain، تمامی تنظیمات اولیه کلی سیستم انجام شده وCPU ها آماده فعالیت هستند. بنابراین، ساختار CPUدر این مرحله کاملاً آماده مقداردهی است.

سوال 21:

با توجه به سیاست‌های پیاده‌سازی شده در سطح‌های دوم و سوم و همچنین استفاده از روش time-slicing، توضیح دهید چرا همچنان مشکل starvation رخ دادن دارد؟

پاسخ:

در سطح دوم، از الگوریتم Shortest Job First (SJF) استفاده می‌شود ودر سطح سوم، از الگوریتم First Come, First Servedاستفاده می‌شود و در سطح اول، از الگوریتم Round Robin استفاده می‌شود.

1. **چرا گرسنگی در SJF رخ می‌دهد؟**

- **SJF**پردازه‌های کوتاه‌تر را ترجیح می‌دهد، و پردازه‌های با زمان اجرای طولانی به‌دلیل ورود مداوم پردازه‌های کوتاه‌تر به صف، ممکن است هرگز اجرا نشوند. این رفتار ذاتاً به گرسنگی پردازه‌های طولانی‌تر منجر می‌شود.

- حتی اگر از روش‌های زمان‌بندی وزنی مانند(WRR) برای تنظیم زمان‌بندی استفاده شود، این مشکل حل نمی‌شود، زیرا WRR تأثیری بر رفتار SJF ندارد.

2. **چرا گرسنگی در FCFS رخ می‌دهد؟**

- **FCFS**پردازه‌ها را بر اساس ترتیب ورود اجرا می‌کند. اگر پردازه‌ای با حلقه بی‌نهایت وجود داشته باشد، CPUرا برای مدت نامحدود اشغال می‌کند و پردازه‌های دیگر نمی‌توانند اجرا شوند.

3. **چرا این مشکلات در Round Robin رخ نمی‌دهد؟**

- **Round Robin**با تخصیص کوانتوم زمانی محدود، پردازه‌ها را به صورت چرخشی اجرا می‌کند. حتی پردازه‌های با حلقه بی‌نهایت نمی‌توانند CPU را برای مدت طولانی اشغال کنند، و همه پردازه‌ها در بازه زمانی معقولی به CPU دسترسی پیدا می‌کنند.

نتیجه :مشکل گرسنگی در **SJF**و **FCFS** ناشی از فقدان مکانیسم‌های کنترل زمانی است، اما **Round Robin**با استفاده از کوانتوم زمانی، از بروز این مشکلات جلوگیری می‌کند.

سوال 22:

به چه علت مدت زمانی که پردازه در وضعیت SLEEPING می‌باشد، به عنوان زمان انتظار پردازه از منظر زمان‌بندی در نظر گرفته نمی‌شود؟

پاسخ:

مدت زمانی که پردازه در حالت SLEEPINGاست، به عنوان زمان انتظار پردازه در نظر گرفته نمی‌شود زیرا:

1. فعال نبودن در صف پردازنده:

- زمانی که پردازه در حالت SLEEPINGقرار دارد، در صف پردازنده (Ready Queue)نیست و منتظر رویداد یا منبع خاصی است. این مدت زمان به زمان‌بند مربوط نمی‌شود، زیرا زمان‌بند تنها پردازه‌های RUNNABLEرا مدیریت می‌کند.

2. وابستگی به منابع خارجی:

- پردازه در حالت SLEEPINGمنتظر رخ دادن یک رویداد خارجی) مثل تکمیل I/O) یا آزاد شدن منبعی است. این زمان به عنوان زمان انتظار در پردازنده در نظر گرفته نمی‌شود، زیرا پردازنده در این مدت، پردازه‌های دیگری را اجرا می‌کند.

3. غیرفعال بودن پردازه:

- در این حالت، پردازه در وضعیت غیرفعال است و نمی‌تواند در فرآیند زمان‌بندی مشارکت داشته باشد. این مدت زمان تنها به عملکرد سیستم و مدیریت منابع خارجی وابسته است.