گزارش پروژه‌ی نهایی درس سیستم‌عامل

استاد جوادی

اعضای گروه:

دانیال حمدی ۹۷۳۱۱۱۱

امیرحسین رجب‌پور ۹۷۳۱۰۸۵

پاییز ۹۹

بخش اول: سوالات در مورد XV6

سوال اول:

تابع **pinit**:

جدول پردازه‌های **xv6** یا همان **ptable**، از یک **spinlock** برای جلوگیری از رخ دادن شرایط مسابقه، و دیگر مشکلات مسئله‌ی ناحیه‌ی بحرانی استفاده می‌کند. تابع **pinit**، تابع **initlock** را صدا می‌زند، که شئ **lock** مختص **ptable** را مقداردهی اولیه می‌کند.

تابع **cpuid**:

شماره پردازنده فعلی را، که حاصل تفاضل آدرس آن از آدرس **cpu** اول است، برمی‌گرداند.

تابع **mycpu**:

شئ پردازنده‌ی فعلی را (از جنس **struct cpu**) برمی‌گرداند. این تابع تنها باید زمانی که وقفه‌ها غیر فعال هستند صدا زده شود.

تابع **myproc**:

این تابع وقفه‌ها را غیرفعال کرده (**pushcli**)، شئ پردازه‌ی فعلی درون پردازنده را می‌گیرد، و سپس دوباره وقفه‌ها را فعال می‌کند* (**popcli**).

* در واقع تابع **popcli** لزوماً وقفه‌ها را فعال نمی‌کند. تابع **pushcli** مقدار عدد **cli** پردازنده را یک واحد افزایش، و تابع **popcli** این مقدار را یک واحد کاهش می‌دهد. در صورتی که مقدار جدید **cli** برابر ۰ باشد، وقفه‌ها فعال خواهند شد.

تابع **allocproc**:

این تابع جدول پردازه‌ها (**ptable**) را پیمایش کرده، تا یک فرآیند **UNUSED** پیدا کند، در صورت وجود، وضعیت آن را به **EMBRYO** تغییر داده، متغیرهایش را مقداردهی اولیه می‌کند، و آن را برمی‌گرداند.

تابع **userinit**:

اولین فرآیند سیستم عامل را راه اندازی می کند؛ حافظه‌ی مورد نیاز را به آن اختصاص داده، رجیسترها را مقدار دهی کرده و وضعیت آن را به **RUNNABLE** تغییر می‌دهد تا هسته‌های پردازنده بتوانند آن را اجرا کنند.

تابع **growproc**:

حافظه‌ی تخصیص یافته به پردازه‌ی فعلی را به میزان عدد **int** ورودی افزایش (یا کاهش) می‌دهد. میزان حافظه‌ی تخصیص یافته‌ی هر پردازه، در متغیر **sz** ساختارش ذخیره شده است.

تابع **fork**:

یک کپی از پردازه‌ی فعلی را به عنوان فرزند ساخته و منابع مورد نیاز را در اختیارش قرار می‌دهد. منظور از کپی این است که تمام متغیرهای ساختار پردازه‌ی پدر، (به جز مواردی مانند **pid** و **parent***) عیناً تکرار می‌شوند. همچنین وضعیت آن را **RUNNABLE** قرار می‌دهد.

تابع **exit**:

پردازه‌ی فعلی را تمام می‌کند، که شامل بستن فایل‌های باز آن پردازه، و تغییر پردازه‌ی پدر فرزندانش به پردازه‌ی ابتدایی (**initproc**) می‌شود. در نهایت وضعیت پردازه‌ی تمام شده به **ZOMBIE** تغییر می‌کند.

باید توجه داشته باشیم که این تابع منابع تخصیص یافته به پردازه را آزاد نمی‌کند؛ برای این کار باید پردازه‌ی پدر، تابع **wait** را صدا بزند.

تابع **wait**:

وقتی یک پردازه این تابع را صدا بزند، تا زمانی که فرزندی داشته باشد، صبر می‌کند تا کار فرزندانش تمام شده و به حالت **ZOMBIE** در بیایند، که در این صورت، منابع تخصیص یافته به آن فرزند را گرفته، و وضعیت آن را به **UNUSED** تغییر می‌دهد. در صورتی که فرزندی باقی نماند، مقدار -۱ را برمی‌گرداند.

تابع **switch_process**:

این تابع **cpu** و آن فرآیندی که باید **cpu** به آن اختصاص داده شود را گرفته و **cpu** را به آن فرآیند اختصاص می‌دهد.

تابع **scheduler**:

طبق الگوریتم انتخاب شده (متغیر **enum schedPolicy policy**)، عمل زمان‌بندی بین پردازه‌ها را انجام می‌دهد. به طور دقیق‌تر این تابع، طبق متغیر **policy**، در یک حلقه‌ی بی‌نهایت، در هر پیمایش پردازه‌ی مناسب را انتخاب کرده، و پردازنده‌اش را آن اختصاص می‌دهد.

تابع **sched**:

پردازنده را از پردازه‌ی در حال اجرا گرفته، و به پردازه‌ی تابع **scheduler** پس می‌دهد.

تابع **yield**:

پردازه‌ی فعلی را به حالت **RUNNABLE** برده، و با فراخوانی تابع **sched**، پردازنده را از آن می‌گیرد.

تابع **forkret**:*

اولین فرزند حاصل از **fork**.

تابع **sleep**:

پردازه‌ی فعلی را به وضعیت **SLEEPING** می‌برد؛ در مواقعی که پردازه‌ی فعلی منتظر ورودی از کاربر، بخشی از منابع سخت‌افزاری، یا ... باشد، این تابع استفاده می‌شود. در این صورت متغیر ***chan** (مخفف ***channel**، که نشان‌دهنده‌ی نوع منبع مورد انتظار است) غیر صفر خواهد بود.

تابع **wakeup1**:

تمام پردازه‌هایی که منتظر رخداد در ***chan** بوده‌اند، را از وضعیت **SLEEPING** به وضعیت **RUNNABLE** می‌برد. (بیدار می‌کند.)

تابع **wakeup**:

قفل جدول پردازه‌ها را گرفت، تابع **wakeup** را فراخوانی می‌کند، و در نهایت قفل را آزاد می‌کند.

تابع **kill**:

این تابع شماره‌ی یک فرآیند را گرفته و آن را **kill** می‌کند. اگر فرآیند در حالت **sleeping** بود آن را به حالت **runnable** می‌برد.

تابع **procdump**:

شماره (**pid**)، وضعیت (**state**)، و نام تمام پردازه‌های موجود را برای خطایابی چاپ می‌کند. برای فرخوانی آن از **P^** استفاده می‌شود.

تابع **updateStateDurations**:

این تابع برای محاسبه ی زمانی است که فرآیند در حالت های مختلف از جمله **sleeping, runnable** و **running** می باشد. درواقع به کمک این زمان ها می توان **turn around time** و **waiting time** و **cpu burst time** را محاسبه کرد.

تابع **getParentID**:

این تابع شماره ی فرآیند پدر آن فرآیندی که این تابع را صدا زده برمی گرداند.

تابع **getChildren**:

این تابع تعداد فرزندان آن فرآیندی که این تابع را صدا زده برمی گرداند.

تابع **getSyscallCounter**:

این تابع نشان می دهد که هر سیستم کال توسط آن فرآیند چند بار صدا زده شده است.

تابع **setPriority**:

این تابع یک عدد بین 1 تا 6 به عنوان اولویت گرفته و آن را اولویت آن فرآیندی که این تابع را صدا زده قرار می دهد.

تابع **changePolicy**:

این تابع سیاست جدید را می‌گیرد و آن را به عنوان سیاست سیستم‌عامل قرار می‌دهد که 0 نشانگر الگوریتم **round robin** و 1 نشانگر الگوریتم **priority scheduling** و 2 نشانگر الگوریتم چند صفی می باشد.

تابع **getTurnAroundTime**:

این تابع شماره‌ی یک پراسس را گرفته و مقدار **turn around time** آن پراسس برمی‌گرداند.

تابع **getWaitingTime**:

این تابع شماره‌ی یک پراسس را گرفته و مقدار **waiting time** آن پراسس برمی‌گرداند.

تابع **getCBT**:

این تابع شماره‌ی یک پراسس را گرفته و مقدار **cpu burst time** آن پراسس برمی‌گرداند.

تابع **customWait**:

این تابع آدرس شروع یک آرایه را می‌گیرد که این آرایه ۴ خانه دارد که خانه‌ی اول آن مقدار **turn around time** و خانه‌ی دوم مقدار **waiting time** و خانه‌ی سوم آن مقدار **cpu burst time** و خانه‌ی چهارم آن اولویت آن فرآیند است.

سوال دوم:

الگوریتم زمان‌بندی پیش‌فرض **6XV**، الگوریتم **Round Robin** به صورت **Pre-emptive**، با میزان **Quantum** = 1 می‌باشد.

برای تغییر این **Quantum** الگوریتم **Round Robin**:

- ابتدا میزان **Quantum** مورد نظر خود را تعریف می‌کنیم. (ترجیحاً در فایل **param.h**)

- درون ساختار پردازه‌ها (**struct proc**)، یک متغیر **rr_remaining_t** اضافه می‌کنیم، که نشان‌دهنده‌ی میزان زمان سپری شده‌ی یک پردازه، از حداکثر زمان مجازش در هر دور (**Quantum**) می‌باشد. (تغییر در فایل **proc.h**)
- تابع **trap** را به گونه‌ای تغییر می‌دهیم که به جای بازپس‌گیری پردازنده از پردازه‌ی فعلی با هر بار رخداد، یک واحد از زمان باقی‌مانده‌ی پردازه کم می‌کنیم، و تنها در صورت صفر شدن آن، پردازنده را از آن می‌گیریم. (تغییر در فایل **trap.c**)

برای تغییر الگوریتم به **Priority scheduling**:

- درون ساختار پردازه‌ها (**struct proc**)، یک متغیر **priority** اضافه می‌کنیم، که نشان‌دهنده‌ی الویت آن پردازه می‌باشد.
- تابع **scheduler** را به گونه‌ای تغییر می‌دهیم که در هر دور، پردازه‌ی در وضعیت **RUNNABLE**ای که بالاترین الویت را دارد برگزیده شود.

سوال سوم:

فراخوانی‌های سیستمی واسطی برای ارتباط سطح **user** با سطح **kernel** می‌باشند. این توابع به کاربر اجازه می‌دهند تا به کمک توابع از پیش تعریف شده، کارهای خود را جلو ببرند.

برای تعریف یک فراخوانی سیستمی

- در قدم اول، در فایل **syscall.h**، نام و شماره فراخوانی سیستمی جدید را تعریف می‌کنیم.
- تابع مورد نظر این فراخوانی را به فایل **sysproc.c** اضافه می‌کنیم.
- در فایل **syscall.c**، تابع مورد نظر را ابتدا **extern** کرده، و سپس به آرایه‌ی **syscalls** اضافه می‌کنیم.
- در صورت نیاز، منطق اصلی تابع را به تابعی در **proc.c** منتقل کرده، و تابع جدید را در تابع قبلی صدا می‌زنیم. (در این صورت باید پروتوتایپ تابع جدید را به فایل **defs.h** هم اضافه کنیم.)

- برای آن که فراخوانی سیستمی، در فضای **User**، قابل دسترسی باشد، تابع را به فایل **user.h** اضافه می‌کنیم.
- در فایل **usys.s**، فراخوانی سیستمی جدید را اضافه می‌کنیم.

هر فراخوانی سیستمی در **6XV**، توسط تابع **syscall** در **syscall.c** صدا زده می‌شود. این تابع شماره‌ی فراخوانی درخواست شده را از روی فیلد **eax** ثبّات **tf** خوانده، و در صورت معتبر بودن شماره، تابع مورد نظر را از روی آرایه‌ی **syscalls** اجرا می‌کند. خروجی این فراخوانی سیستمی دوباره روی فیلد **eax** از ثبّات **tf** نوشته می‌شود.

کد پروژه:

https://github.com/amirhoseinRj/Operating-System-Project