

به نام خدا

پروژه فاز ۱ درس سیستم‌های عامل

مدرس:

دکتر رضا انتظاری

منتور پروژه:

امیرمحمد سهرابی

اعضای گروه:

نوید ابراهیمی

سینا علی‌نژاد


آذر ۱۴۰۱

توضیح کد:

۱- فایل syscall.h:

برای معرفی سیستم کال و اضافه کردن شماره‌ای که بعداً در یک آرایه استفاده می‌شود از فایل استفاده می‌کنیم. در این فایل اسم system call که باید برای این پروژه اضافه می‌کردیم (SYS_proc_dump) را اضافه می‌کنیم.

```
h syscall.h
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define SYS_sleep 13
15 #define SYS_uptime 14
16 #define SYS_open 15
17 #define SYS_write 16
18 #define SYS_mknod 17
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_proc_dump 22
24
```



عکس شماره ۱- اضافه کردن دستور مورد نظر

۲- فایل syscall.c:

در این فایل آرایه syscalls داریم که با استفاده از شماره‌هایی که در فایل syscall.h تعریف کردیم، این آرایه را مقداردهی میکنیم و مقدارهای آن پوینترهایی به systemcallها هستند.


```
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 extern int sys_proc_dump(void);
107
108 static int (*syscalls[])(void) = {
109     [SYS_fork]    sys_fork,
110     [SYS_exit]    sys_exit,
111     [SYS_wait]    sys_wait,
112     [SYS_pipe]    sys_pipe,
113     [SYS_read]    sys_read,
114     [SYS_kill]    sys_kill,
115     [SYS_exec]    sys_exec,
116     [SYS_fstat]   sys_fstat,
117     [SYS_chdir]   sys_chdir,
118     [SYS_dup]     sys_dup,
119     [SYS_getpid]  sys_getpid,
120     [SYS_sbrk]    sys_sbrk,
121     [SYS_sleep]   sys_sleep,
122     [SYS_uptime]  sys_uptime,
123     [SYS_open]    sys_open,
124     [SYS_write]   sys_write,
125     [SYS_mknod]   sys_mknod,
126     [SYS_unlink]  sys_unlink,
127     [SYS_link]    sys_link,
128     [SYS_mkdir]   sys_mkdir,
129     [SYS_close]   sys_close,
130     [SYS_proc_dump] sys_proc_dump,
131 };
132
```

عکس شماره ۲- اضافه کردن دستور مورد نظر

۲- فایل defs.h:

در این قسمت function prototype را به defs.h که یک header فایل است اضافه میکنیم. این کار را در سکشن proc.c انجام میدهیم (در مرحله ۱ این دستور را تعریف کردیم)

```
106 int      cpuid(void);
107 void      exit(void);
108 int      fork(void);
109 int      growproc(int);
110 int      kill(int);
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void      pinit(void);
114 void      procdump(void);
115 void      scheduler(void) __attribute__((noreturn));
116 void      sched(void);
117 void      setproc(struct proc*);
118 void      sleep(void*, struct spinlock*);
119 void      userinit(void);
120 int      wait(void);
121 void      wakeup(void*);
122 void      yield(void);
123 | int      proc_dump(void);
```



عکس شماره ۳- اضافه کردن دستور مورد نظر

۳- فایل user.h:

در این فایل همانند مورد دوم، function prototype را این بار به user.h اضافه میکنیم.

۴- فایل usys.S:

در این فایل همانند مورد دوم، function prototype را این بار به usys.S اضافه میکنیم.

۵- فایل sysproc.c:

در این فایل number سیستم کالی که این number در syscall.h تعریف شده است را return میکند. این function در syscall.c صدا زده شده بود.

```
93 | int sys_proc_dump(void)
94 | {
95 |     return proc_dump();
96 | }
97 |
```

عکس شماره ۴- تعریف این فانکشن در این فایل

۶- فایل proc.c:

در این فایل تابع proc_dump را تعریف میکنیم. در ابتدا روی آرایه‌ای که ارائه دهنده سیستم‌کال‌ها هست acquire میزنیم تا تمام process‌هایی که در حالت running یا runnable هستند را به آرایه processes اضافه میکنیم. وقتی اینکار انجام شد با release میزنیم تا lock این قسمت برداشته شود. سپس این process‌ها را sort میکنیم. به این صورت که ابتدا بر اساس sort memsize میکنیم و در ادامه اگر memsize برای دو process برابر بود، آن دو را براساس sort id میکنیم.

در آخر کار آرایه process‌ها را که sort کردیم چاپ میکنیم. عدد ۲۲ که نشان‌دهنده proc_dump هست را نیز برمیگردانیم.

```

543     {
544         struct proc_info processes[NPROC];
545         int l = 0;
546         struct proc_info *pr = processes;
547         struct proc *p;
548         acquire(&ptable.lock);
549         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
550         {
551             if(p->state == RUNNING || p->state == RUNNABLE)
552             {
553                 struct proc_info info = {p->pid, p->sz};
554                 *pr = info;
555                 pr++;
556                 l++;
557             }
558         }
559         release(&ptable.lock);
560         for (int i = 0; i < l; ++i)
561         {
562             for (int j = i + 1; j < l; ++j)
563             {
564                 if (processes[i].memsize > processes[j].memsize)
565                 {
566                     struct proc_info a = processes[i];
567                     processes[i] = processes[j];
568                     processes[j] = a;
569                 }
570                 else if (processes[i].memsize == processes[j].memsize)
571                 {
572                     if (processes[i].pid > processes[j].pid)
573                     {
574                         struct proc_info a = processes[i];
575                         processes[i] = processes[j];
576                         processes[j] = a;
577                     }
578                 }
579                 else {
580                     continue;
581                 }
582             }
583         }
584         for (int i = 0; i < l; ++i)
585         {
586             if (processes[i].pid != 0)
587             {
588                 cprintf("pid: %d, memsize: %d\n", processes[i].pid, processes[i].memsize);
589             }
590         }
591         return 22;

```

عکس شماره 5- تابع proc_dump

۷- فایل MakeFile:

در این قسمت اسم دستور را به UPROGS و اسم Function را به EXTRA اضافه میکنیم.

```

251 EXTRA=\
252     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
253     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c proc_dump.c zombie.c\
254     printf.c umalloc.c\
255     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
256     .gdbinit.tmpl gdbutil\

```

عکس شماره 6- اضافه کردن function دستور به EXTRA

```

168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _proc_dump\
184     _zombie\

```

عکس شماره ۷- اضافه کردن اسم دستور به UPROGS

۸- فایل proc_dump.c:

ابتدا یک آرایه تعریف میکنیم که اعداد رندم در آن جای دارند که این برای اختصاص دادن سایزهای مختلف به processهایی که بعدا تشکیل میدهیم می باشد. داخل یک حلقه به تعداد طول آرایه process جدید میسازیم که malloc را در این قسمت انجام میدهیم. (1) While جهت تبدیل نشدن process به zombie است زیرا با اینکار child process تمام نمیشود و مشغول میماند. در ادامه تابع proc_dump را صدا میزنیم.

```

5  int
6  main(int argc, char const *argv[]) {
7      int arr[14] = {200, 500, 1200, 1500, 2000, 5000, 8000, 10000, 12000, 15000, 20000, 50000, 80000, 100000};
8      int num = sizeof(arr)/sizeof(arr[0]);
9      for (int i = 0; i < num; i++)
10     {
11         int r = fork();
12         if (r == 0)
13         {
14             int* b = malloc(arr[i] * sizeof(int));
15             *b = 0;
16             while (1)
17                 continue;
18             exit();
19         }
20     }
21     proc_dump();
22     exit();
23 }

```

عکس شماره ۸- پیاده سازی proc_dump

مشکلات پیاده‌سازی:

۱- نبود `printf`: برای اینکار باید از دستور `cprintf` استفاده میکردیم زیرا در سطح `kernel` دستور `printf` وجود ندارد.

۲- نبود تابع `wait` کتابخانه C: در سطح `kernel` دستور `wait(NULL)` وجود ندارد و بجای آن باید از تابع `wait`ی که در خود سیستم عامل تعریف شده است استفاده کرد.

۳- تشکیل `zombie`: برای رفع این مشکل از `while(1)` استفاده کردیم.