به نام خدا

گزارش پروژه دوم xv6

محمد جواد زندیه 9831032

در این گزارش فایل هایی که تغییر داده شده اند به همراه بخش هایی که تغییر یافته است را بررسی می کنیم:

1. Proc.c

برای انجام کارهای مختلف نیاز است تا به Race condition هم دقت شود و به همین منظور برای ترد هایمان یک قفل تعریف می کنیم.

```
18  struct spinlock thread;
```

```
30    initlock(&thread, "thread");
```

در هنگام allocate کردن یک پردازه باید به thread, stackTop هم مقدار بدهیم زیرا در فایل proc.h و در PCB مربوط به پردازه ها این دو ویژگی جدید را اضافه کرده ایم، مقدار دهی اولیه می کنیم:

```
96    p->stackTop = -1; // initialize stackTop to -1 (illegal value)
97    p->threads = -1; // initialize threads to -1 (illegal value)
```

```
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 readid;                  // keep acount for read calls
  int threads;                 // number of threads
  int stackTop;                // top of stack for this process
};
```

وقتی که یک پردازه را ایجاد می کنیم در ابتدا فقط خودش به عنوان ترد این پردازه می باشد:

```
140   // only one thread is executing for this process
141   p->threads = 1;
```

در هنگام گسترش فضای آدرس یک پردازه باید فضای آدرس ترد های آن نیز افزایش یابد در صورت وجود چون فضای ترد و پدرش یکسان است.

```
173   acquire(&thread);
174   sz = curproc->sz;
175   if(n > 0){
176     if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0){
177       release(&thread);
178       return -1;
179     }
180
181   } else if(n < 0){
182     if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0){
183       release(&thread);
184       return -1;
185     }
```

```
189   curproc->sz = sz;
190   acquire(&ptable.lock);
191   // we should update sz in all threads due to existence of thread_create system call
192   struct proc *p;
193   int numberOfChildren;
```

```
190   acquire(&ptable.lock);
191   // we should update sz in all threads due to existence of thread_create system call
192   struct proc *p;
193   int numberOfChildren;
194   // check if it is a child or parent
195   if(curproc->threads == -1) // child
196   {
197     // update parents sz
198     curproc->parent->sz = curproc->sz;
199     // -2 is because of update parent along with one child
200     numberOfChildren = curproc->parent->threads - 2;
201     if(numberOfChildren <= 0){
202       release(&ptable.lock);
203       release(&thread);
204       switchuvm(curproc);
205       return 0;
206     }else{
207       for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
208         if(p!=curproc && p->parent == curproc->parent && p->threads == -1){
209           p->sz = curproc->sz;
210           numberOfChildren--;
211         }
212       }
213     }
214   }
```

```
215     else{ // is not a child
216       numberOfChildren = curproc->threads - 1;
217       if(numberOfChildren <= 0){
218         release(&ptable.lock);
219         release(&thread);
220         switchuvm(curproc);
221         return 0;
222       }
223       else{
224         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
225           if(p->parent == curproc && p->threads == -1){
226             p->sz = curproc->sz;
227             numberOfChildren--;
228           }
229         }
230       }
231     }
232
233     release(&ptable.lock);
234     release(&thread);
235     switchuvm(curproc);
236     return 0;
237   }
```

ترد ها دارای stackTop یکسان با پدر خود هستند و وقتی fork می کنیم یک ترد اضافه می شود پس:

```
262     // child has the same stack top as parent in fork
263     np->stackTop = curproc->stackTop;
264     // there is only one thread for the child because in fork
265     np->threads = 1;
```

اگر تردی Exit را صدا بزند نباید بلافاصله فضای آدرس را free کنیم چون ممکن است ترد های دیگری در حال استفاده از این فضای آدرس باشند پس باید دقت شود که آیا تردی دیگری در حال استفاده از آن هست یا نه

```
318     // if a child calls exit, decrement the number of threads sharing the same pgdir for parent
319     if(curproc->threads == -1){
320       curproc->parent->threads--;
321     }
```

```
341   int
342   check_pgdir_share(struct proc *process)
343   {
344     struct proc *p;
345     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
346       if(p != process && p->pgdir == process->pgdir)
347         return 0;
348     }
349     return 1;
350   }
```

Thread_wait روی ترد های فرزند عمل خواهد کرد در wait باید اگر ترد فرزند نیست wait کنیم و بالعکس پس:

```
369         // only thread_wait handles child threads
370         if(p->threads == -1)
371           continue;
```

همچنین برای free کردن هم می بایست چک شود که آخرین ترد باشیم:

```
379         if(check_pgdir_share(p))
380           freevm(p->pgdir);
```

چون در تابع wait مربوط به پردازه هستیم و نه ترد پس باید threads آنرا -1 قرار دهیم

```
387         // reset stackTop and pgdir if it is the parent
388         p->stackTop = -1;
389         p->pgdir = 0;
390         p->threads = -1;
```

حال به پیاده سازی thread_create می رسیم:

ابتدا پردازه کنونی که myproc هست را پیدا کرده و یک پردازه جدید از آن allocate می کنیم

سپس تعداد ترد های پردازه curproc را یک واحد افزایش می دهیم چون قرار است پردازه جدید به عنوان ترد این پردازه باشد

سپس stackTop پردازه و ترد آنرا یکسان کرده و فضای هر دو را یکسان می کنیم

```
673 int
674 thread_create(void *stack)
675 {
676   int pid;
677   // curproc is the current process
678   struct proc *curproc = myproc();
679   // np is the new process
680   struct proc *np;
681   // allocate process
682   if((np = allocproc()) == 0)
683     return -1;
684
685   // increase threads number for parent, default value of threads for child is -1
686   curproc->threads++;
687
688   // Remember stack grows downwards Thus the stackTop will be in the address given be parent
689   np->stackTop = (int)((char*)stack + PGSIZE);
690   // might be at the middle of changing address space in another thread
691   acquire(&ptable.lock);
692   np->pgdir = curproc->pgdir;
693   np->sz = curproc->sz;
694   release(&ptable.lock);
695
696   int bytesOnStack = curproc->stackTop - curproc->tf->esp;
697   np->tf->esp = np->stackTop - bytesOnStack;
```

تعداد بایت هایی که در استک باید ذخیره شود را بدست می آوریم و تمام آنرا از پدر در ترد فرزند کپی می کنیم

```
696   int bytesOnStack = curproc->stackTop - curproc->tf->esp;
697   np->tf->esp = np->stackTop - bytesOnStack;
698   memmove((void*)np->tf->esp, (void*)curproc->tf->esp, bytesOnStack);
699
700   np->parent = curproc;
701
702   // copying all trapframe register values from curproc(parent) into np(child)
703   *np->tf = *curproc->tf;
704
705   // clear eax so that fork returns 0 in the child
706   np->tf->eax = 0;
707
708   // esp points to the top of the stack (esp is the stack pointer)
709   np->tf->esp = np->stackTop - bytesOnStack;
710   // ebp is the base pointer
711   np->tf->ebp = np->stackTop - (curproc->stackTop - curproc->tf->ebp);
712
713   int i;
714   for(i = 0; i < NOFILE; i++)
715     if(curproc->ofile[i])
716       np->ofile[i] = filedup(curproc->ofile[i]);
717   np->cwd = idup(curproc->cwd);
718
719   safestrcpy(np->name, curproc->name, sizeof(curproc->name));
```

```
720
721   pid = np->pid;
722
723   acquire(&ptable.lock);
724
725   np->state = RUNNABLE;
726
727   release(&ptable.lock);
728
729   return pid;
730
731 }
```

حال در thread_wait هم داریم که:

```
733 int
734 thread_wait(void)
735 {
736   struct proc *p;
737   int havekids, pid;
738   struct proc *curproc = myproc();
739
740   acquire(&ptable.lock);
741   for(;;){
742     // Scan through table looking for exited children.
743     havekids = 0;
744     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
745       if(p->parent != curproc)
746         continue;
747       if(p->threads != -1) // remember thread_wait only waits for threads not child processes
748         continue;
749       havekids = 1;
750       if(p->state == ZOMBIE){
751         // Found one.
752         pid = p->pid;
753         kfree(p->kstack);
754         p->kstack = 0;
755
756         if(check_pgdir_share(p))
757           freevm(p->pgdir);
```

```
756         if(check_pgdir_share(p))
757           freevm(p->pgdir);
758
759         p->pid = 0;
760         p->parent = 0;
761         p->name[0] = 0;
762         p->killed = 0;
763         p->state = UNUSED;
764         p->state = 0;
765         p->pgdir = 0;
766         p->threads = -1;
767         release(&ptable.lock);
768         return pid;
769       }
770     }
771
772     // No point waiting if we don't have any children.
773     if(!havekids || curproc->killed){
774       release(&ptable.lock);
775       return -1;
776     }
777
778     // Wait for children to exit.  (See wakeup1 call in proc_exit.)
779     sleep(curproc, &ptable.lock);  //DOC: wait-sleep
780   }
781 }
```

باید دقت شود که در اینجا فقط برای ترد ها wait می کنیم و نه پردازه ها

2. حال باید دو سیستم کال جدیدرا در سیستم عامل تعریف کنیم و مشابه فاز یک عمل می کنیم:

Syscall.c

```
112  extern int sys_thread_create(void);
113  extern int sys_thread_wait(void);
```

```
140  [SYS_thread_create] sys_thread_create,
141  [SYS_thread_wait] sys_thread_wait
```

Defs.h

```
126  int             thread_create(void* stack);
127  int             thread_wait(void);
```

Syscall.h

```
26  #define SYS_thread_create 25
27  #define SYS_thread_wait 26
```

Sysproc.c

```
111  int
112  sys_thread_create(void){
113    void* stack;
114    if(argptr(1, (void*)&stack, sizeof(*stack)) < 0)
115      return -1;
116    return thread_create(stack);
117  }
118
119  int
120  sys_thread_wait(void){
121    return thread_wait();
122  }
```

User.h

```
29  int thread_create(void* stack);
30  int thread_wait(void);
```

Usys.S

```
35  SYSCALL(thread_create)
36  SYSCALL(thread_wait)
```

3. حال باید یک تابع پوششی به عنوان thread_creator بسازیم:

در این تابع یک ترد جدید ایجاد می شود و آی دی آن باز گردانده می شود.

```c
#include "types.h"
#include "stat.h"
#include "user.h"

#define PAGESIZE 4096

int
thread_creator(void (*fn) (void *), void *arg)
{
    // allocating 2 * pageSize for fptr in heap
    void *fptr = malloc(2 * PAGESIZE);
    void *stack;

    if(fptr == 0)
        return -1;

    int mod = (uint)fptr % PAGESIZE;

    // the following if-else is for assigning page-aligned space to statck
    if(mod == 0)
        stack = fptr;
    else
        stack = fptr + (PAGESIZE - mod);

    int thread_id = thread_create((void*)stack);

    // thread create failed
    if(thread_id < 0)
        printf(1, "thread create failed\n");

    // child
    else if(thread_id == 0){
        // call the function passed to thread_create
        (fn)(arg);
        // free space when function is finished
        free(stack);
        exit();
    }
    return thread_id;

}
```

این تابع را باید در Makefile و user.h معرفی کنیم:

```c
int thread_creator(void (*fn) (void *), void *arg);
```

```
ULIB = ulib.o usys.o printf.o umalloc.o thread_creator.o
```

4. همچنین باید stackTopرا در فایل exec.c مقداری دهی اولیه کنیم

```c
curproc->stackTop = sp;
```

5. سپس باید تست بنویسیم و در Makefile هم این تست ها را بشناسانیم به xv6

```c
#include "types.h"
#include "stat.h"
#include "user.h"

void childPrint(void* args){
    printf(1, "hi childes function executed properly with argument : %d\n", *(int*)args);
}

int main(void){
    int argument = 0x0F01; // 3841 decimal
    int thread_id = thread_creator(&childPrint, (void*)&argument);
    if(thread_id < 0)
        printf(1, "thread_creator failed");

    thread_wait();

    printf(1, "thread_id : %d\n", thread_id);

    exit();
}

```

این تست بالا برای تست کردن درسی عملکرد تابع پوششی thread_creator می باشد

در تست زیر هم داریم:

```c
// threads.c is for testing
#include "types.h"
#include "stat.h"
#include "user.h"

int stack[4096] __attribute__ ((aligned (4096)));
int x = 0;

int main(int argc, char *argv[]){
    printf(1, "Stack is at %p\n", stack);
    // int tid = fork();
    int tid = thread_create(stack);

    if(tid < 0){
        printf(2, "error!\n");
    } else if (tid == 0){
        // child
        for(;;){
            x++;
            sleep(100);
        }
    } else{
        //parent
        for(;;){
            printf(1, "x = %d\n", x);
            sleep(100);
        }
```

```
28      }
29
30      exit();
31  }
```

حال در Makefile تعریف می کنیم تست هایمان را:

```
255 EXTRA=\
256     mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
257     ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
258     printf.c umalloc.c\
259     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
260     .gdbinit.tmpl gdbutil\
261     sysTest.c\
262     getProcCountTest.c\
263     getReadCountTest.c\
264     threadsTest1.c\
265     threadsTest.c\
```

```
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     _zombie\
184     _sysTest\
185     _getProcCountTest\
186     _getReadCountTest\
187     _threadsTest\
188     _threadsTest1\
```

6. در نهایت خروجی:

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 17824
echo           2 4 16676
forktest       2 5 9596
grep           2 6 20044
init           2 7 17264
kill           2 8 16704
ln             2 9 16556
ls             2 10 19188
mkdir          2 11 16804
rm             2 12 16780
sh             2 13 29416
stressfs       2 14 17692
usertests      2 15 68808
wc             2 16 18556
zombie         2 17 16372
sysTest        2 18 16380
getProcCountTe 2 19 16404
getReadCountTe 2 20 16404
threadsTest    2 21 17004
threadsTest1   2 22 21060
console        3 23 0
$ threadsTest
hi childes function executed properly with argument : 3841
thread_id : 5
```

```
$ threadsTest
hi childes function executed properly with argument : 3841
thread_id : 5
$ threadsTest1
Stack is at 2000
x = 0
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
x = 10
x = 11
x = 12
```