به نام خدا

فاز ۲ پروژه سیستم عامل

محمد چوپان ۹۸۳۱۱۲۵

ابتدا تابع clone را برای ساختن یک پردازه جدید تعریف میکنیم.

با این تفاوت که یک ریسمان است. و متفاوت است به شکل زیر پیاده سازی می کنیم.

```
int
clone(void* stack){
  int pid;
  //define current procees
  struct proc *curproc = myproc();
  //define new process
  struct proc *newproc;
  //define new process's stack
  if((newproc = allocproc()) == 0){
    cprintf("allocproc failed\n");
    return -1;
  }
  //increas ethread number for parent defoult is -1
  curproc->threads++;
  // stack top
  newproc->stackTop = (int)((char*)stack+PGSIZE);
  acquire(&ptable.lock);
  newproc->pgdir=curproc->pgdir;
  newproc->sz=curproc->sz;
  release(&ptable.lock);

  int bytesOnStack = curproc->stackTop - curproc->tf->esp;
  newproc->tf->esp = newproc->stackTop - bytesOnStack;
  memmove((void*)newproc->tf->esp, (void*)curproc->tf->esp, bytesOnStack);

  newproc->parent = curproc;

  // copying all trapframe register values from p into newp
  *newproc->tf = *curproc->tf;

  // Clear %eax so that fork returns 0 in the child.
  newproc->tf->eax = 0;

  // esp points to the top of the stack (esp is the stack pointer)
  newproc->tf->esp = newproc->stackTop - bytesOnStack;
```

```
int i;
for(i = 0; i < NOFILE; i++)
  if(curproc->ofile[i])
    newproc->ofile[i] = filedup(curproc->ofile[i]);
newproc->cwd = idup(curproc->cwd);

safestrcpy(newproc->name, curproc->name, sizeof(curproc->name));

pid = newproc->pid;

acquire(&ptable.lock);

newproc->state = RUNNABLE;

release(&ptable.lock);

return pid;

}
```

سپس بجای wait از join استفاده میکنیم. ولی با این تفاوت که تنها باید ریسمان ها را از بین ببرد و نه پردازه ها را به همین دلیل تابعی دیگر را تعریف میکنیم که این مساله را چک کند.

```
//check page directory table
int
check_pgdir_share(struct proc * process){
  struct proc *p;
  for(p=ptable.proc;p<&ptable.proc[NPROC];p++){
    if(p!= process &&p->pgdir==process->pgdir){
      return 0;
    }
  }
  return 1;
}
```

این تابع درون تمامی پردازه ها حرکت کرده و آنانی که page directory مشترک دارند یعنی ریسمان هستند را تشخیص می دهد.

سپس از این در join & wait استفاده میکنیم تا تفاوت ریسمان و پردازه را تشخیص دهد.

```
// function like wait
// just wait for threads
// return 0 on success, -1 on failure
// wait for all threads to finish
int
join(void){
  struct proc* p;
  int havekids, pid;
  struct proc *curproc = myproc();
  acquire(&ptable.lock);
  //test
  if(curproc->threads == 0){
    release(&ptable.lock);
    return -1;
  }
  for(;;){
    // Scan through table looking for exited children.
    havekids = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->parent != curproc)
        continue;

      if(p->threads!=-1){//just wait for threads
        continue;
      }
      havekids = 1;
      if(p->state == ZOMBIE){
        // Found one.
        pid = p->pid;
        kfree(p->kstack);
        p->kstack = 0;
        //check if no threads free page directory
        if(check_pgdir_share(p)==0){
          freevm(p->pgdir);
        }
```

```
        p->pid = 0;
        p->parent = 0;
        p->name[0] = 0;
        p->killed = 0;
        p->state = UNUSED;
        release(&ptable.lock);
        return pid;
      }
    }

    // No point waiting if we don't have any children.
    if(!havekids || curproc->killed){
      release(&ptable.lock);
      return -1;
    }

    // Wait for children to exit.  (See wakeup1 call in proc_exit.)
    sleep(curproc, &ptable.lock);  //DOC: wait-sleep
  }

}
```

سپس شرط زیر را به تابع wait نیز اضافه میکنیم.

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

در قسمت init هم این دو را اضافه میکنیم

```
static struct proc *initproc;
struct spinlock thread;
int nextpid = 1;
extern void forkret(void);
extern void trapret(void);

static void wakeup1(void *chan);

void
pinit(void)
{
  initlock(&ptable.lock, "ptable");
  initlock(&thread, "thread");
}
```

تا بتوان thread را نیز لاک کرد.

Struct proc را نیز به شکل زیر تغییر می دهیم و متغیر stack &thread را نیز به آن ها اضافه میکنیم.

```
  int threads;                // Number of threads using this process and same memory
  int stackTop;               // Top of stack
};
```

سپس باید این تغییرات را اعمال کنیم و یک فایل جدید ایجاد میکنیم تا از این توابع استفاده کرده و ریسمان ما را بسازد.

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#define PAGESIZE 4096

//thread create function
int thread_create(void (*fn) (void*),void* arg){
    void *fptr = malloc(2 * (PAGESIZE));
    void *stack;
    if(fptr == 0){
        printf(1,"thread_create: out of memory\n");
        return -1;

    }
    int mod=(uint)fptr%PAGESIZE;
    if(mod!=0){
        stack=fptr+PAGESIZE-mod;
    }
    else{
        stack=fptr;
    }
    int thread_id=clone((void*)stack);
    //check the CLONE
    if(thread_id<0){
        printf(1,"thread_create: clone failed\n");
        return -1;
    }
    //child
    else if(thread_id==0){
        //call the function
        fn(arg);
        //fre the stack
        free(stack);
        //exit
        exit();
    }
    return thread_id;
```

این تابع ریسمان ما را می سازد .

تغییرات این فایل ها را درون فایل های دیگر اعمال میکنیم همانند فاز ۱

در نهایت فایل های تست خود را می سازیم.

البته در آخر نیز فایل makefile را تغییر می دهیم.

```
getProcCountTest.c\
getReadCountTest.c\
threads.c\
thereadsTest.c\
thread_test.c\
test_thread.c\
```

```
_zombie\
_getProcCountTest\
_getReadCountTest\
_threads\
_thereadsTest\
_thread_test\
_test_thread\
```

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

فایل:threads.c

```c
//want to test thread create
#include "types.h"
#include "stat.h"
#include "user.h"
//do therads and  for share memory and fork
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 = clone(stack);

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

  exit();
}
```

در اینجا حافظه اشتراکی برای ریسمان ها تعریف میکنیم و خروجی زیر را میگیریم.

```
ls              2 10 19116
mkdir           2 11 16732
r               2 12 16708
s               2 13 29344
stressfs        2 14 17624
usertests       2 15 68720
wc              2 16 18480
zombie          2 17 16292
getProcCountTe  2 18 16384
getReadCountTe  2 19 16384
threads         2 20 20964
thereadsTest    2 21 16928
thread_test     2 22 18380
test_thread     2 23 17928
console         3 24 0
$ threads
Stack is at 2000
x = 0
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
```


حال اگر کد را به شکل زیر تغییر دهیم.

```
mkdir           2 11 16732
rm              2 12 16708
sh              2 13 29344
stressfs        2 14 17624
usertests       2 15 68720
wc              2 16 18480
zombie          2 17 16292
getProcCountTe  2 18 16384
getReadCountTe  2 19 16384
threads         2 20 16840
thereadsTest    2 21 16928
thread_test     2 22 18380
test_thread     2 23 17928
console         3 24 0
$ threads
x = 0
x = 0
x = 0
x = 0
x = 0
x = 0
x = 0
x = 0
x = 0
x = 0
x = 0
```

همانطور که میبینید برای پردازه حافظه اشتراکی پاسخ نمی دهد.

فایل therheadsTest.c

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

void childPrint(void* args){
    printf(1, "hi, childs function executed properly with argument : %d\n", *(int*) args);
}
//just create thread
int main(void){
    int argument = 0x0F21;
    int thread_id = thread_create(&childPrint, (void*)&argument);
    if(thread_id < 0)
        printf(1, "thread_create failed\n");

    join();

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

    exit();
}
```

wait

خروجی:

```
threads         2 20 16840
thereadsTest    2 21 16928
thread_test     2 22 18380
test_thread     2 23 17928
console         3 24 0
$ thereadsTest
hi, childs function executed properly with argument : 3873
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inod
estart 32 bmap start 58
init: starting sh
$
```

همان طور که مشاهده میکنید ریسمان ساخته شده و خروجی مطلوب را چاپ می کند.

فایل test_thread.c

```c
void
testThread(int n)
{
  int tid, i;
  for (i = 0; i < n; i++) {
    tid = thread_create(run, &i);
    sleep(10);
    if (tid < 0) {
      printf(1, "create thread failed\n");
    } else {
      printf(1, "create thread %d\n",i+1);
      sleep(10);
    }
  }

  // join
  for (i = 0; i < 5 ; i++) {
    tid = join();
  }
  printf(1, "all threads joined\n");
}

int
main(int argc, char *argv[]){
    testThread(3);
    exit();
}
```

```c
#include "types.h"
#include "stat.h"
#include "user.h"
//run function to test thread and calcultion with threads
void
run(void* arg)
{
  int id = *(int*) arg;
  id++;
  int i;
  sleep(100);
  for( i = 0 ; i < 4 ; i++){
    printf(1, "This is thread %d printing %d\n", id, i);
    if( id != 2 || i != 3)
      sleep(100);
    if (id == 1 && i == 1)
      sleep(100);
  }
  printf(1, "Thread %d finished execution\n", id);
  exit();
}
```

در این فایل محاسباتی را با ریسمان انجام میدهیم.

```
nit: starting sh
 test_thread
reate thread 1
reate thread 2
reate thread 3
his is thread 1 printing 0
his is thread 2 printing 0
his is thread 3 printing 0
his is thread 1 printing 1
his is thread 2 printing 1
his is thread 3 printing 1
his is thread 2 printing 2
his is thread 3 printing 2
his is thread 1 printing 2
his is thread 2 printing 3
hread 2 finished execution
v6...
pu1: starting 1
pu0: starting 0
b: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inod
start 32 bmap start 58
nit: starting sh
```

۳ ریسمان را در تابع ابتدایی ساخته و با آن ها داده ها را چاپ میکنیم و مبینیم که شرایط مسابقه ایجاد نمی شود.

فایل :thread_test.c

```
pid = fork();
if (pid < 0) {
  printf(1, "create process failed\n");
} else if (pid == 0) {
  printf(1, "fork: child\n");
  exit();
} else {
  printf(1, "create process (id = %d)\n", pid);
  if (wait() != pid) {
    printf(2, "create_n_threads: wait unexpected process (expected = %d)\n", pid);
  }
}

// join
for (i = 0; i < threadCounter; i++) {
  tid = join();
  printf(1, "tid = %d join \n", tid);
}
}


nt
ain(int argc, char *argv[])

 create_n_threads(100);
 exit();
```

```
#include "types.h"
#include "stat.h"
#include "user.h"
void
sleep_func(void* arg)
{
  printf(1, "welcome to sleep_func\n");
  int time = *(int*)arg;
  sleep(time);
  exit();
}

//function do test thread and fork to see pgtable and join and wait
void
create_n_threads(int n)
{
  printf(1, "welcome to test_n_threads\n");
  int tid, i, pid;
  int threadCounter = 0;
  int time = 100;
  for (i = 0; i < n; i++) {
    tid = thread_create(sleep_func, &time);
    sleep(10);
    if (tid < 0) {
      printf(1, "create thread failed\n");
    } else {
      printf(1, "create thread (id = %d)\n", tid);
      threadCounter++;
    }
  }
```

در این فایل هم برای پردازه و ریسمان جدول صفحه آن را مشاهده و تست میکنیم.

```
threadstest    2 21 16928
thread_test    2 22 18380
test_thread    2 23 17928
console        3 24 0
$ thread_test
welcome to test_n_threads
welcome to sleep_func
create thread (id = 5)
welcome to sleep_func
create thread (id = 6)
welcome to sleep_func
create thread (id = 7)
cfroerak:t ech iprlocde
ss (id = 8)
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 log
estart 32 bmap start 58
```