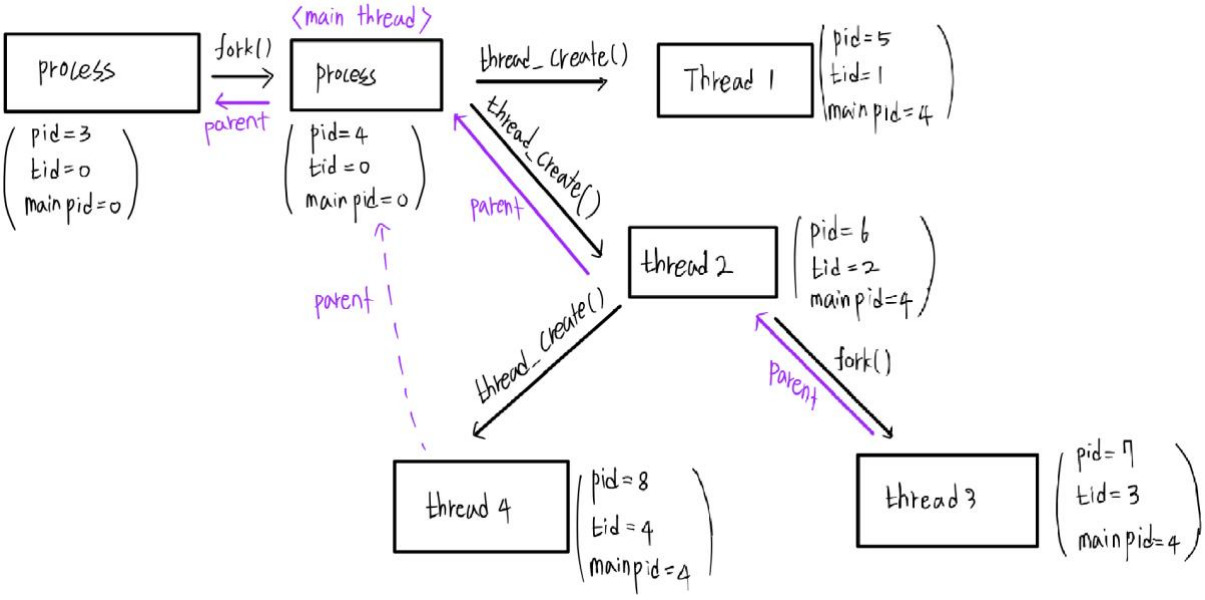
Operation System Assignment 2 Wiki

2017033654 류동원

1. 디자인



A. 메인 스레드에서 thread_create를 진행할 때

   i. 각 스레드는 메인 스레드의 pid를 mainpid에 저장함

   ii. 각 스레드는 메인 스레드를 parent로 가짐

   iii. pid와 tid는 생성된 순서대로 할당

B. 일반 스레드에서 thread_create를 진행할 때

   i. 각 스레드는 메인 스레드의 pid를 mainpid에 저장

   ii. 각 스레드는 메인 스레드를 parent로 가짐

C. 일반 스레드에서 fork를 진행할 때

   i. 각 스레드는 fork를 호출한 일반스레드를 parent로 가짐

   ii. 각 스레드는 메인 스레드의 pid를 mainpid에 저장

2. 구현

A. proc.h 파일의 proc 구조체의 추가 멤버

```
/* pmanager */
int limit;                  // process memory limit
int stackPageNum;           // number of stack pages

/* thread */
uint tid;                    // Thread ID
int mainpid;                 // 메인 쓰레드의 pid 저장 (자신이 메인쓰레드이면 0)
void *retval;                // return value
struct proc *tparent;        // 쓰레드가 쓰레드를 create 한 경우
```

i. limit : 프로세스의 메모리 limit

ii. stackPageNum : 프로세스의 스택 수

iii. tid : Thread ID

iv. mainpid : 메인 쓰레드의 pid 저장

v. tparent : 쓰레드가 쓰레드를 create한 경우 부모 쓰레드의 값

vi. retval : return value

B. exec2 함수 (proc.c 파일)

i. 스택용 페이지

1. exec 함수를 기반으로 가드용 페이지 1개 + user stack용 페이지 stacksize개를 할당받도록 수정한다.

2. stacksize의 수를 1이상 100이하의 정수로 제한한다.

```
// 스택용 페이지를 여러 개 할당받을 수 있도록 변경
// Make the first inaccessible.  Use the second as the user stack.
if (stacksize > 100 || stacksize < 1)
  goto bad;
sz = PGROUNDUP(sz);

// stack용 페이지 할당 : 가드용 1개 + stack용 stacksize개
if((sz = allocuvm(pgdir, sz, sz + (stacksize+1)*PGSIZE)) == 0)
  goto bad;
clearpteu(pgdir, (char*)(sz - (stacksize+1)*PGSIZE));
sp = sz;
```

3. 프로세스의 stakPageNum 값 변경해준다.

```
// 프로세스의 stackPageNum 값 변경
curproc->stackPageNum = stacksize + 1 + curproc->stackPageNum;
```

C. setmemorylimit 함수 (proc.c 파일)

1. ptable을 search하며 pid가 일치하는 프로세스를 찾는다. 찾지 못하면 -1 리턴
2. 입력된 limit 값과 프로세스의 limit 값이 0이면 변경없이 0 리턴
3. 입력된 limit 값이 0이고 프로세스의 limit 값이 0이 아니면 프로세스의 limit 값을 0으로 할당 및 0 리턴
4. 입력된 limit값이 음수이면 -1 리턴
5. 기존의 프로세스 사이즈보다 입력된 limit 값이 작을 경우 -1 리턴

```
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->pid == pid){
    // limit이 0인 경우 제한이 없음
    if (p->limit == 0 && limit == 0) {
      release(&ptable.lock);
      return 0;
    } else if (p->limit != 0 && limit == 0) {
      p->limit = 0;
      release(&ptable.lock);
      return 0;
    // limit이 음수인 경우
    } else if (limit < 0) {
      release(&ptable.lock);
      return -1;
    // 기존 할당받은 메모리보다 limit이 작은 경우
    } else if (limit < p->sz) {
      release(&ptable.lock);
      return -1;
    }
    // limit이 양수인 경우
    p->limit = limit;
    release(&ptable.lock);
    return 0;
  }
}
// pid가 존재하지 않는 경우
release(&ptable.lock);
return -1;
```

6. 입력된 limit이 양수이며 프로세스의 사이즈보다 큰 값이 정상적으로 입력되면 프로세스의 limit 값을 변경 후 0 리턴

D. Process manager (pmanager.c 파일에 구현)

i. getcmd 함수 : 명령어를 읽고 buf 변수에 저장, 성공적으로 읽었으면 0을 리턴

ii. getnext 함수 : 명령어를 읽어 저장된 buf에서 각각의 argument값을 구하고 리턴

iii. strcmp2 : 명령어에 대응하는 숫자를 리턴함

1. { list=1, kill=2, execute=3, memlim=4, exit=5, default=-1 };

iv. neg_atoi 함수 : 문자형 변수를 int로 변경 (음수도 변경)

v. main함수의 while loop

1. while문이 돌면서 매번 명령어를 읽고 switch문으로 명령어에 대응하는 처리 구문으로 이동

```
while(getcmd(buf, sizeof(buf)) >= 0) {
    int num = strcmp2(buf);
    // num { list=1, kill=2, execute=3, memlim=4, exit=5, default=-1 };
    switch (num) {
        case 1:
            list_process();
            break;
        case 2:
            arg1 = getnext(buf, newbuf, 1);
            kill(atoi(arg1));
            break;
```

2. case 1 : list

   A. 현재 실행 중인 프로세스들의 정보를 출력한다.

   B. list_process() 시스템 콜 함수 호출 -> proc.c 파일에 구현

```
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if (p->state == RUNNABLE || p->state == RUNNING || p->state == SLEEPING) {
    // procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
    // tid == 0 -> Thread의 경우 출력하지 않는다.
    if(p->tid == 0) {
      cprintf("pid %d\t %s\t %d\t stackPageNum %d\t size %d\t memoryLimit %d tid %d\
          p->pid, p->name, p->state, p->stackPageNum, p->sz, p->limit, p->tid);
    }
  }
}
```

3. case 2 : kill <arg1>

   A. kill 시스템 콜 호출

4. case 3 : execute <arg1> <arg2>

   A. fork 이후, 자식 프로세스가 exec2 시스템 콜 호출

B. 부모 프로세스는 별도로 wait 하지 않음

```
case 3:
    arg1 = getnext(buf, newbuf, 1);
    arg2 = getnext(buf, newbuf2, 2);
    char *argv[] = { arg1 };
    if(fork1() == 0) {
        if(exec2(arg1, argv, atoi(arg2)) == -1) {
            printf(2, "프로그램 실행에 실패하였습니다.\n");
        }
    }
    break;
```

5. case 4 : memlim <arg1> <arg2>

   A. setmemorylimit 시스템 콜 함수 호출 -> proc.c 파일에 구현

```
case 4:
    arg1 = getnext(buf, newbuf, 1);
    arg2 = getnext(buf, newbuf2, 2);
    if(setmemorylimit(atoi(arg1), neg_atoi(arg2)) == 0) {
        printf(2, "memory limit 설정에 성공하였습니다.\n");
    }
    break;
```

6. case 5 : exit

   A. exit() 함수 호출

7. default

   A. 명령어를 다시 입력받음

3. Light-weight process

   E. thread_create 함수 (proc.c 파일에 구현)

      i. 초기화

         1. mainpid는 메인스레드의 pid
         2. parent는 메인스레드의 process 값
         3. tparent는 현재 생성하고 있는 스레드의 process 값
         4. tid는 스레드가 생성될 때마다 1씩 증가시키면서 할당

```
if(curproc->tid == 0) { // curproc이 메인 쓰레드일 경우
  np->mainpid = curproc->pid;
  np->parent = curproc;
  np->tparent = curproc;
} else if (curproc->tid > 0) { // curproc이 일반 쓰레드일 경우
  np->mainpid = curproc->mainpid;
  np->parent = curproc->parent;
  np->tparent = curproc;
}
np->tid = nexttid++;
*thread = np->tid;
```

ii. 스레드는 부모의 pgdir을 공유하며, 가드 스택과 유저 스택 각각 1개씩 별도의 스택을 추가로 가진다.

iii. parent의 프로세스 사이즈를 할당 받고 증가된 스텍 크기만큼 사이즈를 최신화

```
// 프로세스 페이지 테이블 공유
np->pgdir = np->parent->pgdir;
*np->tf = *curproc->tf;
np->sz = np->parent->sz;
np->parent->sz += 2*PGSIZE;
// 쓰레드 스택 할당
if((np->sz = allocuvm(np->pgdir, np->sz, np->sz + 2*PGSIZE)) == 0)
    np->state = UNUSED;
    return -1;
}
```

iv. 현재 프로세스의 trap frame을 공유한다.

v. argument와 레지스터 값을 할당해준다.

```
ustack[0] = 0xffffffff;
ustack[1] = (uint)arg;
np->tf->esp = np->sz - 8;
if(copyout(np->pgdir, np->tf->esp, ustack, 8) != 0) {
  return -1;
}
np->tf->eip = (uint)start_routine;
```

F. thread_exit 함수 (proc.c 파일에 구현)

i. retval 값을 저장해준 후, 자신의 상태는 zombie로 만들고 부모 프로세스를 wakeup 한다.

```
curproc->retval = retval; // 리턴값 저장

acquire(&ptable.lock);
wakeup1(curproc->parent);
curproc->state = ZOMBIE;
sched();
panic("zombie exit\n");
```

G. thread_join 함수 (proc.c 파일에 구현)

i. wait 함수와 유사하게 for문을 돌면서 인자로 넘겨받은 thread값과 동일한 tid를 가진 프로세스를 찾는다.

ii. 동일한 프로세스를 찾았으면 flg = 1로 변경하고 프로세스가 zombie 상태이면 자원을 회수한다.

iii. 동일한 프로세스를 찾았으나 zombie 상태가 아니라면 sleep하고 자식 프로세스가

종료될 때까지 기다린다.

```
for(;;) {
    flg = 0;
    for(p=ptable.proc; p<&ptable.proc[NPROC]; p++) {
        if(p->tid != thread)
            continue;
        flg = 1;
        if(p->state == ZOMBIE) {
          *retval = p->retval;
          // free up basic resources
          kfree(p->kstack);
          p->kstack = 0;
          p->pid = 0;
```

iv. 동일한 프로세스가 없었으면 -1을 리턴한다.

```
    }
    // 동일한 tid가 없을 경우
    if(!flg) {
      release(&ptable.lock);
      return -1;
    }
    sleep(curproc, &ptable.lock);
}
```

H. allocproc 함수

i. 프로세스의 변수들을 초기화한다.

```
found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->limit = 0;
  p->stackPageNum = 0;
  p->tid = 0;
  p->mainpid = 0;
  p->tparent = 0;
  release(&ptable.lock);
```

I. growproc 함수

i. 추가적인 메모리를 할당 받을 때 프로세스의 메모리 limit을 넘지 않도록 함.

```
if (curproc->limit != 0 && curproc->limit < sz + n) {
  cprintf("[memory limit 초과] 추가적인 메모리 할당이 불가능합니다.");
  return -1;
}
```

J. fork 함수

i. 스레드에서 fork가 호출된 경우 tid를 할당해주고 parent는 현제 스레드, mainpid는 메인스레드의 pid로 초기화해준다.

```
if(curproc->tid > 0) { // thread에서 호출되었을 때 thread 추가
  np->tid = nexttid++;
  np->parent = curproc;
  np->mainpid = curproc->mainpid;
  np->tparent = curproc;
} else {              // 일반 프로세스에서 호출되었을 경우
  np->parent = curproc;
}
```

K. wait 함수

i. 자식 프로세스가 좀비 상태이면 종료시켜 줄 때 thread 관련 메모리도 초기화

```
p->stackPageNum = 0;
p->limit = 0;
p->mainpid = 0;
p->tid = 0;
p->retval = 0;
```

L. kill 함수

i. 스레드가 kill 되었을 경우 같은 메인스레드에서 생성된 모든 스레드를 종료시켜 준다.

```
// kill 하려는 프로세스를 찾아서 curproc에 저장
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->pid == pid) {
    curproc = p;
  }
}
```

```
// thread가 kill 되었을 때 같은 메인스레드의 thread도 kill
if (curproc->tid > 0) {
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if (p->tid > 0) {
      if(p->mainpid == curproc->mainpid) {
        p->killed = 1;
        if (p->state == SLEEPING)
          p->state = RUNNABLE;
      }
    }
  }
  release(&ptable.lock);
  return 0;
```

ii. 메인 스레드(프로세스)가 kill 되었을 경우, 메인 스레드 뿐만 아니라 메인 스레드에서 생성된 스레드들을 모두 kill 한다.

```
} else {  // main process가 kill 되었을 때
  for(pt = ptable.proc; pt < &ptable.proc[NPROC]; pt++){
    if( (pt->tid > 0) && (pt->mainpid == curproc->pid) ) {
      pt->killed = 1;
      if(pt->state == SLEEPING)
        pt->state = RUNNABLE;
    }
  }
  curproc->killed = 1;
  if(curproc->state == SLEEPING)
    curproc->state = RUNNABLE;
  release(&ptable.lock);
  return 0;
}
```

M. clear_thread 함수 (proc.c 파일에 구현)

i. exec 함수 호출 시 메인 스레드이면 메인 스레드에서 생성된 스레드들 모두 종료

```
if(curproc->tid == 0) {
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if ((p->tid > 0) && (curproc->pid && p->mainpid)) {
      p->killed = 1;
      if(p->state == SLEEPING) {
        p->state = RUNNABLE;
      }
    }
  }
```

ii. 일반 스레드일 경우 자신의 스레드를 제외한 나머지 메인 스레드에서 생성된 스레드 종료

```
} else { // 일반 스레드일 경우 자신의 스레드를 제외한 나머지 스레드 종료
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if ((p->tid > 0) && (curproc->pid && p->mainpid)) {
        if(curproc != p){
          p->killed = 1;
          if(p->state == SLEEPING) {
            p->state = RUNNABLE;
          }
        }
      }
    }
  }
}
```

N. exec 함수

i. exec 함수 실행할 때 프로세스의 스레드 정리

```
clear_thread();
```

4. 기타

A. defs.h 파일

i. 새로 생성한 함수 선언

```
int             setmemorylimit(int pid, int limit);
int             exec2(char *path, char **argv, int stacksize);
void            list_process(void);
int             thread_create(thread_t *thread, void* (*start_routine)(void*), void* arg);
void            thread_exit(void* retval);
int             thread_join(thread_t thread, void** retval);
void            clear_thread(void);
```

B. types.h 파일

i. thread_t 변수 선언

```
typedef uint thread_t;
```

C. usys.S 파일

i. system call 함수

```
SYSCALL(exec2)
SYSCALL(setmemorylimit)
SYSCALL(list_process)
SYSCALL(thread_create)
SYSCALL(thread_exit)
SYSCALL(thread_join)
SYSCALL(clear_thread)
```

D. user.h 파일

i. system call 함수

```
int exec2(char*, char**, int);
int setmemorylimit(int, int);
void list_process(void);
int thread_create(thread_t*, void*, void*);
void thread_exit(void*) __attribute__((noreturn));
int thread_join(thread_t, void**);
void clear_thread(void);
```

E. proc_syscall.c 파일

i. 시스템 콜의 wrapper function 함수 구현

F. Makefile 파일

```
proc_syscall.o\
prac_syscall.o\
```

```
_pmanager\
_my_app\
_thread_exec\
_thread_exit\
_thread_kill\
_thread_test\
```

```
EXTRA=\
    mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
    ln.c ls.c mkdir.c rm.c stressfs.c wc.c zombie.c\
    printf.c umalloc.c pmanager.c mlfq_test.c my_app.c\
    thread_exec.c thread_exit.c thread_kill.c thread_test.c\
```

G. syscall.c 파일

```
[SYS_exec2]   sys_exec2,
[SYS_setmemorylimit]  sys_setmemorylimit,
[SYS_list_process]  sys_list_process,
[SYS_thread_create] sys_thread_create,
[SYS_thread_exit]   sys_thread_exit,
[SYS_thread_join]   sys_thread_join,
[SYS_clear_thread]  sys_clear_thread
extern int sys_exec2(void);
extern int sys_setmemorylimit(void);
extern int sys_myfunction(void);
extern int sys_list_process(void);
extern int sys_thread_create(void);
extern int sys_thread_exit(void);
extern int sys_thread_join(void);
extern int sys_clear_thread(void);
```

H. syscall.h 파일

```
#define SYS_exec2  23
#define SYS_setmemorylimit  24
#define SYS_list_process    25
#define SYS_thread_create   26
#define SYS_thread_exit     27
#define SYS_thread_join     28
#define SYS_clear_thread    29
```

5. Test

i. Pmanager.c test

```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ pmanager
$$ list
pid 1    init    2       stackPageNum 2  size 12288      memoryLimit 0 tid 0
pid 2    sh      2       stackPageNum 2  size 16384      memoryLimit 0 tid 0
pid 3    pmanager        4       stackPageNum 2  size 16384      memoryLimit 0 tid 0
$$ memlim 3 20000
memory limit 설정에 성공하였습니다.
$$ list
pid 1    init    2       stackPageNum 2  size 12288      memoryLimit 0 tid 0
pid 2    sh      2       stackPageNum 2  size 16384      memoryLimit 0 tid 0
pid 3    pmanager        4       stackPageNum 2  size 16384      memoryLimit 20000 tid 0
```

ii. thread_kill test

```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ thread_kill
Thread kill test start
Killing procesThis code shouldThis code should be executed!This cThis code should s 4
 be executed!!
!
ode shoube executed!!
This code ld be executed!!
should be executed!!zombie!

Kill test finished
```

iii. thread_test test (test 3 제외하고 테스트 진행)

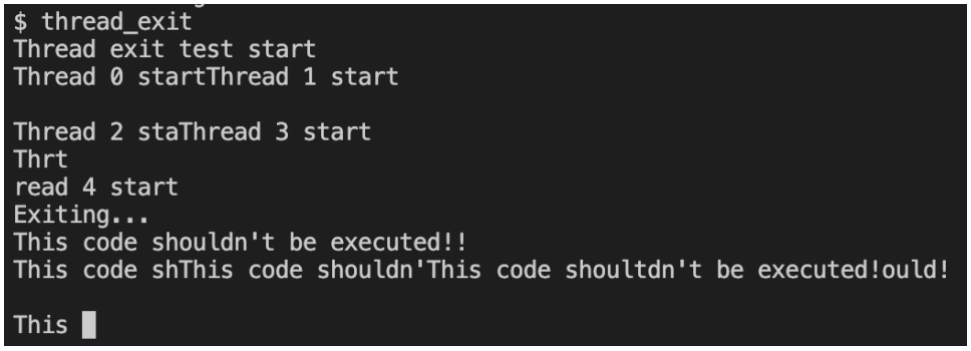
```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00


Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ thread_test
Test 1: Basic test
create_all 호출
Thread 0 start
Thread 0 end
Thread 1 start
Parent waiting for children...
Thread 1 end
Test 1 passed

Test 2: Fork test
create_all 호출
TThreadThreadThread hrea 2 sta3 start
d 1 start
rt
Thread 4 start
 0 start
Child of thread Child of thread 3 starChild of thread 2 start
Child of threadt
1 st 4 start
Child of threart
ad 0 start
Child of thrChild of thread 2 eend
Threaad 3 end
Child of thread 1 Child of thread Child of thread 0 d 2 end
end
4 end
endThread Thread
Thr3 end
4 end
Thread 0 end
ead 1 end
Test 2 passed
```

iv. thread_exit 테스트



6. Trouble shooting

A. sbrk는 구현하지 못함 -> Thread_test 파일의 테스트에서 제외

B. thread_kill 파일에서 좀비 프로세스가 발생함

i. 부모 프로세스가 종료할 때 자식프로세스가 존재하면 parent를 init process로 할당하는데 왜 이런 문제가 발생하는지 해결하지 못함.

C. thread_exit 테스트는 테스트파일과 명세가 미세한 차이로 인해 정상작동하지 않음

i. exit 호출시 프로세스 내 모든 스레드가 종료되도록 구현하지 않음

D. thread_exec 파일은 오류 해결하지 못함.