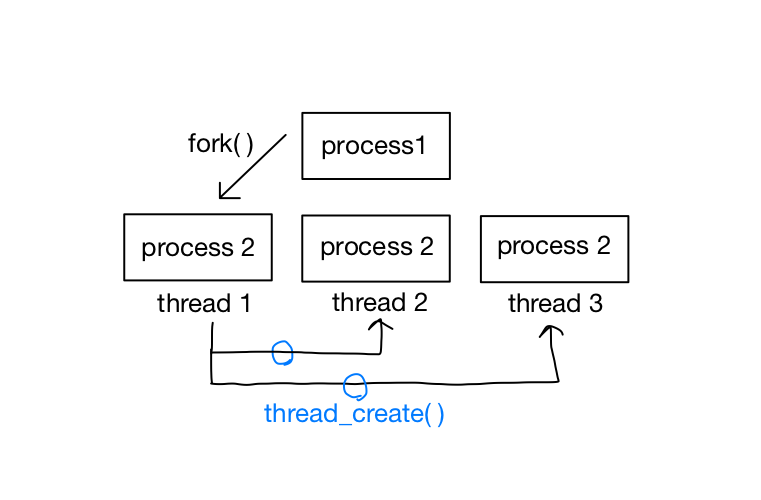
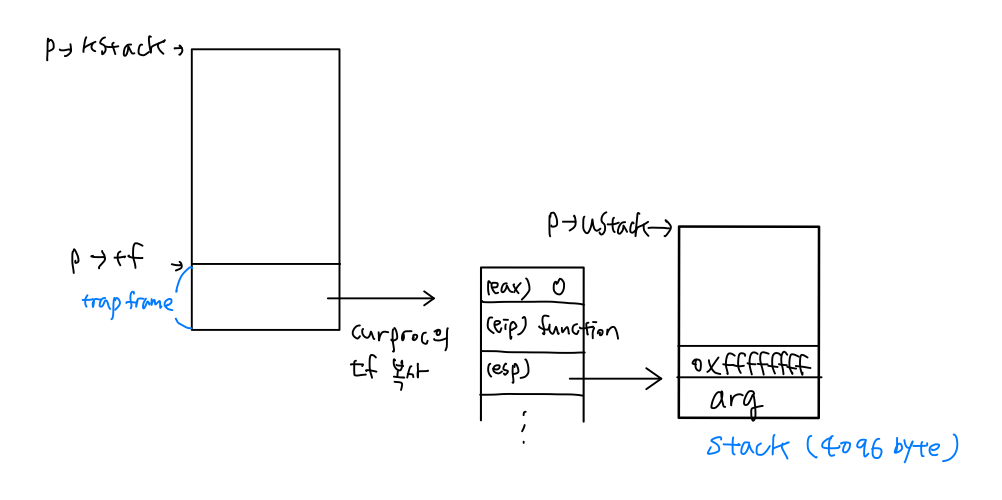
<OS Project 2> 2015310884 박소현

1. thread\_create

thread\_create( )의 역할을 다음의 표로 그려보았다.



쓰레드에 메모리를 할당해 주는 방식은 다음과 같이 표현할 수 있다.



<proc.h>

- proc구조체에 다음 변수들을 추가

struct proc{

...

char \*ustack;

int isthread;

int tid; //real tid

int for\_tid; //for giving tid not related with thread count

int thread\_count; //count for threads, parent가 이 정보를 소유

void \*returnvalue; //thread\_exit 과 thread\_join에서 쓰인다.

...

}

<proc.c>

int thread\_create(void \*(\*function)(void \*), void \*arg, void \*stack)

- struct proc \*proc = myproc(); struct proc \*np;

- np에 프로세스를 할당한다.

- acquire(&ptable.lock);

- proc->thread\_count > 8 이면, release(&patble.lock)하고 return -1

- pgdir, sz, parent, pid, nicevalue, tf를 parent process로부터 그대로 복사해온다.

- np->tf->eax = 0 으로 %eax를 clear해준다.

- file을 연다.

- np->ustack에 인자로 들어오는 stack을 할당해준다.

- np->state는 RUNNABLE 로 할당해주고, 이것이 thread인지 구분해주는 is thread를 1로 지정해준다.

-쓰레드를 처음 만드는 상황이라면(proc->thread\_count == 0),

proc->thread\_count = 2; proc->for\_tid = 2; 로 할당해준다.

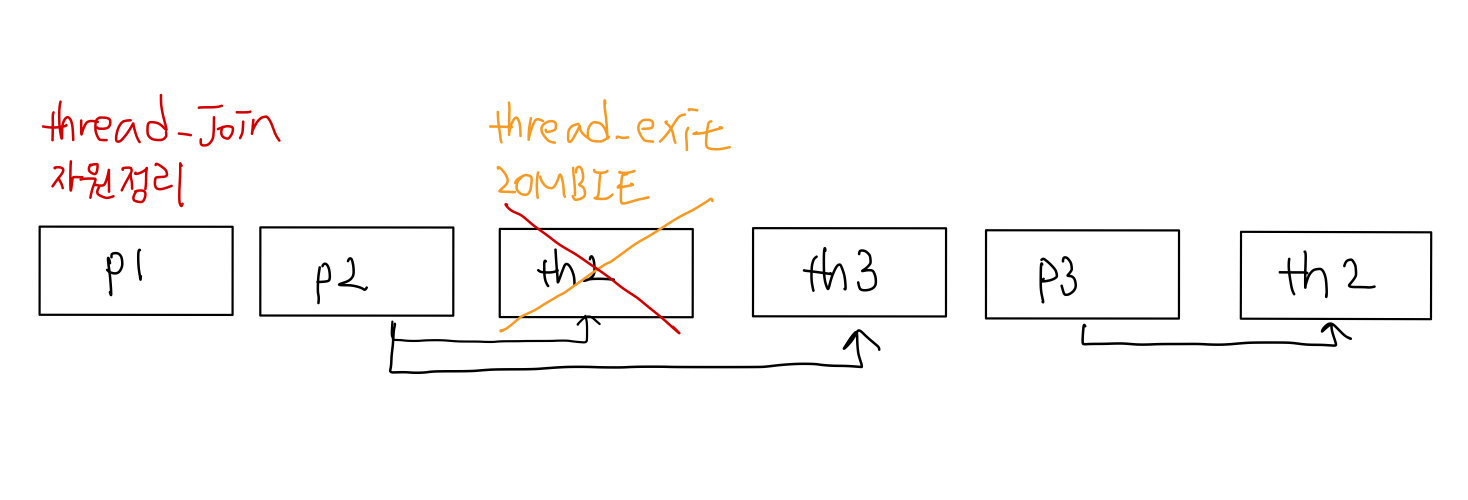
그렇지 않다면,

proc->thread\_count와 proc->for\_tid를 1씩 증가시킨다.

- np->tid = proc->for\_tid; 넣어주고, int tid = np->tid; 하여 tid를 return한다.

2. thread\_exit 과 thread\_join

먼저, thread\_exit과 thread\_join의 작동 방식을 그림으로 표현해보았다.



<proc.c>

void thread\_exit(void \*retval)

- struct proc \*proc = myproc();

- acquire(&ptabel.lock);

- proc->returnvalue에 인자로 들어온 retval를 저장한다.

- 파일을 close 한다.

- proc->state를 ZOMBIE로 만든다.

- thread의 parent를 wakeup한다.

int thread\_join(int tid, void \*\*retval)

- struct proc \*proc = myproc();

- acquire(&ptable.lock);

for(;;) {

- ptable의 process를 하나씩 돌며,

p->tid == tid이고, p->pid == p->parent->pid이고, 좀비인 상태를 발견하면,

- p->returnvalue를 \*retval에 넣어준다.

- 파일 close

- kstack을 해제하고 값들을 0으로 초기화하며 자원을 정리한다.

- release(&ptable.lock);

- 성공시 return 0;

-특정thread가 exit을 할 수 있도록 proc이 기다려줘야한다.

sleep(proc, &patable.lock);

}

- release(&ptable.lock);

- 실패시 return -1;

3. gettid

-현재 프로세스의 tid를 반환하는 것은 간단하게 구현할 수 있다.

<proc.c>

int gettid(void){

struct proc \*proc = myproc();

int tid = proc->tid;

return tid;

}

4. exit( ) 수정

- 프로세스 상태일 때(p->isthread == 0 && p->parent == curporc),

p->parent = initproc; 으로 할당해주고,

p->state == ZOMBIE라면, initproc을 깨운다. (wakeup1)

- 쓰레드 상태일 때, curproc과 같은 pid의 process 들은 모두

-file close 당하고.

-p->state는 UNUSED로 바뀌고,

-kstack은 해제된다.

5. 주의할 점

(1)

<sysproc.c>

void sys\_thread\_exit(void)에서argument를 int로 받아주어야한다.

void \*retval라고 prt로 받으면, proc.c 파일에서 인자 전달이 제대로 이루어지지 않는다. 그래서 다음과 같이 써주어야 한다.

argint(0, (void\*)&retval);

(2) acquire panic 에러가 고쳐지지 않아서 며칠 동안 삽질을 하였다. acqurie과 release를 쓸 때는 다른 lock에 영향을 끼치지는 않는지, 끼친다면 어떤 영향을 끼치고 있는지 반드시 생각을 하며 사용하여야한다. 이번에는 파일을 닫는 부분에 등장하는 코드begin\_op(); iput(proc->cwd); end\_op(); 에서 lock 문제가 있음을 발견하였다. 위 함수들에서 이미 acquire을 하고 있었기에, release를 따로 해주지 않아 에러가 났던 것이다. 이는 deadlock 상태를 유발하고 있었기에, 매번 실행할 때마다 조금씩 다른 결과를 보여서 원인을 찾기 더 어려웠다.