HW2 20143144433 이영숙 운영체제

In this assignment, I will make thread_create, thread_exit, thread_join and gettid function. I was very confused with the difference between thread and process at the beginning because I can't distinguish them since we can use the almost same format with fork, wait, exit. When I learned thread and process from the lecture, the properties of each notion was obvious. While I was doing this assignment, I figured out that those functions are likely to be seen as similar ones but the biggest difference was 'usage of the stack'. Threads within a process locate in the same stack. On the other hand, processes spend different stack. If we want to create a thread, we have to allocate them in the same stack, meaning we have to give them the same stack address with the parent process.

Even though I couldn't reach to the final stage, I just wanted to show my effort I put here.

1. **Thread_create**

```
int
thread_create(void*(*function)(void*), void *arg, void *stack)
{
      int i, n=0;
      struct proc *nth;
      struct proc *curproc = myproc();
      void *stackarg, *stackret;


      //Allocate process.
      if((nth=allocproc()) == 0){
        return -1;
      }
      /*
      // Copy process state from proc.
      if((nth->pgdir = copyuvm(curproc->pgdir, curproc->sz)) == 0){
        kfree(nth->kstack);
        nth->kstack = 0;
        nth->state = UNUSED;
        return -1;
      }
      */

      nth->pgdir = curproc->pgdir;
      nth->sz = curproc->sz;
      nth->parent = curproc;
      nth->tid = nexttid;
      nexttid++;
      *nth->tf = *curproc->tf;

      nth->tf->eax = 0;

      nth->tf->esp=(int)stack;   //stack pointer
      nth->tf->eip=(int)function;  //function start address

```

Process and thread are sharing page table(nth->pgdir = curproc->pgdir).

Esp has the next pointer. Stack will be assigned to the esp. Function address is in eip.

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

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



acquire(&ptable.lock);
for(nth=ptable.proc;nth<&ptable.proc[NPROC];nth++){
  if(nth->pid==curproc->pid){
    n++; //the number of thread in the same process
  }

  if(8<n)
    return -1; //maximum 8
}

nth->state = RUNNABLE;
release(&ptable.lock);


return nth->tid;
```

According the lab session, tid will be unique in the process. However, I just give every thread the unique tid so that it is easier to be distinguished.

2. Thread_join

Honestly, I can't understand what 'retval' means here. I sent an email to TA but it was not enough to see how this pointer variable works in this case.

```
int
thread_join(int tid, void **retval)
{
  struct proc *nth;
  int havekids, pid;
  struct proc *curproc = myproc();

  pid=0;
  acquire(&ptable.lock);

  for(;;){
    // Scan through table looking for exited children.
    havekids = 0;
    for(nth = ptable.proc; nth < &ptable.proc[NPROC]; nth++){
      if(nth->parent != curproc || nth->pgdir != curproc->pgdir || nth->tid == curproc->tid)
        continue;
      havekids = 1;
      if(nth->state == ZOMBIE){
        pid = nth->pid;
        kfree(nth->kstack);
        nth->kstack = 0;
        freevm(nth->pgdir);
        nth->pid = 0;
        nth->parent = 0;
        nth->name[0] = 0;
        nth->killed = 0;
        nth->pgdir=0;
        nth->state = UNUSED;
        release(&ptable.lock);
        return 0;
      }
    }

    // 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
  }
}
끼워넣기 --
```

3. **Thread_exit**

```
void
thread_exit(void *retval)
{

  struct proc *cth = myproc();
  struct proc *p;
  int fd;

  if(cth == initproc)
      panic("init exiting");

  // Close all open files.
     for(fd = 0; fd < NOFILE; fd++){
       if(cth->ofile[fd]){
         fileclose(cth->ofile[fd]);
             cth->ofile[fd] = 0;
        }
     }

  begin_op();
  iput(cth->cwd);
  end_op();
  cth->cwd = 0;

  cth->tf->eax=(int)retval;
  acquire(&ptable.lock);

  // Parent might be sleeping in wait().
  wakeup1(cth->parent);

  // Pass abandoned children to init.
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
     if(p->parent == cth){
        p->parent = initproc;
        if(p->state == ZOMBIE)
           wakeup1(initproc);
      }
  }


   // Jump into the scheduler, never to return.
   cth->state = ZOMBIE;
끼워넣기 --
```

4. **Gettid**

HW2 20143144433 이영숙 운영체제

```
int
gettid(void)
{
  struct proc *curproc = myproc();
    return curproc->tid;
}
```