SUSTech CS302 OS Lab4 Report

Title: Scheduler

Student: Name: Lu Ning, ID: 11610310

Time: 2019. 3. 26

Experimental Environment: linux

Deadline: 11:59, 2018-03-27

Summit by: Blackboard

Task:

Task 1. Create a directory named with your studentID and begin the following task.

Task 2. Find out all bugs in source codes

Task 3. Modify the source code according to the requirements of the experimental instructions, compile and run each program

Task 4. Observe the results and finish the questions

Experiments:

1. Fundamental:

- What is a process? What is a program? And what is the difference?

  Process is an execution of certain program with its own address space. Program is a set of instructions to carry out a specified task.

  Difference: A program is a passive entity, while a process is an active entity. A program becomes a process when a executable code is loaded in the memory. There could be several processes associated with the same program.

- What is job? And what is the difference between process and job?

  Job is a series of program submitted to operating system for some goals.

  Process is a execution of one program, job is a set of programs.

- What are the differences between job scheduling and process scheduling?

  Job scheduling is to choose processes from the job pool and load them to memory.

  Process scheduling is about choosing one process and allocate a CPU for it.

- What are the similarities and differences among structures, classes, and unions in C language：

  Similarity:

- They are user-defined data types to store data of different types as a single unit.
- Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.
- Can be used as input and return value.

Different:

- Field data is pubic in struct and union and private in class, by default.
- Union cannot be inherited.
- Union only contain a value at any given time.

2. Knowledge:

- How many states are in a job? And what are they? READY RUNNING DONE
  job.h

- What programs are used in this experiment for job control?
  schedule.c enq.c deq.c
  And their function?

  void schedule();//调度函数

  void sig_handler(int sig, siginfo_t *info, void *notused);//信号处理

  int allocjid();//分配作业 id

  void do_enq(struct jobinfo *newjob, struct jobcmd enqcmd);//入队函数

  void do_deq(struct jobcmd deqcmd);//出队函数

  void do_stat();//显示作业状态

  void updateall();//更新所有作业信息

  struct waitqueue* jobselect();//等待队列中选择作业

  void jobswitch();//作业转换

- What is used for inter-process communication in this experiment? signal and named pipe (a temporary file)
  And its function?
  kill(int pid, signal)

raise(signal)
open(file, arg)
read()

3. Questions:

- What should be noted when printing out the job name:

  We should use the first element of job -> cmdarg, which is a 2-dimension char array.

- Submit a job that takes more than 100 milliseconds to run:

job4_sleep.c commond ./job4 <sleep time>

```c
// Sleep in c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>


int main(int argc,char *argv[]) {

if(argc == 2)
    {
        char **end;
        int sleepTime = strtol(argv[1], end, 10);
        printf("I will sleep for %d s\n", sleepTime);
        sleep(sleepTime);
        printf("Woke up!!\n");
        return 0;
    }
    else
    {
        printf("error! format: job4 one_integer\n");
        return 0;
    }

}
```

- List the bugs you found and your fix (code and screenshot)

100ms is not correctly set:

```c
interval.tv_sec = 0;
interval.tv_usec = 100000; //bug: usec is microsecond
```

Remove the selected job from the wait queue is not right. If we choose the first job in the queue, the head will be null.(The condition **prevp == p** means selected job is the first)
I add a waitqueue_length to record the queue length.

```
// move the selected job from the wait queue
waitqueue_length--;
if(waitqueue_length == 0)
{
  head = NULL;
  #ifdef DEBUG
    printf("choose the only job JID = %d\n remove the se
  #endif

}
else if(head == select)
{

  head = select->next;

}
else
{
  selectprev->next = select->next;

}
select->next = NULL;
```

If there are two jobs, one is running, the original code will switch to another job after a time slice no matter which job's priority is higher.
I modify the **selectnext()**

```
/* no need to switch because the current job has he highest priority */
if(current)
{
  if(select->job->curpri < current->job->defpri)
  {
    #ifdef DEBUG
      printf("select jid=%d, curpri=%d current jid=%d, defpri=%d\nNo need to swit
      select->job->jid, select->job->curpri, current->job->jid, current->job->def
    #endif

    return NULL;
  }
}
```

Not handle the case where no switch is needed. For example, there is only one job in running and no job in wait queue. Or the running job has the highest priority, the next job to be executed should also be it. Modify the **jobswitch()**

```
} else {    /* next == NULL && current != NULL, no switch */
  kill(current->job->pid, SIGCONT);
  #ifdef DEBUG
  printf("next == NULL && current != NULL \ncurjid = %d ",current->job->jid);
  #endif
  return;
}
```

- Run the job scheduler program and analyze the execution of the submitted job:

```
luning@luning-laptop ~/workspace/OS/lab4 ./enq job4 20
luning@luning-laptop ~/workspace/OS/lab4 ./enq -p 1 job4 10
luning@luning-laptop ~/workspace/OS/lab4 ./enq -p 2 job4 5
```

Jid=1: sleep 20s default priority = 0

Jid=2: sleep 10s default priority = 1

Jid=3: sleep 5s default priority = 2

First execute job1.

Then job2 comes, then execute job2, job1 will be waiting and priority will be added, then job1 and job2 have same priority, job1 will run due to RR mechanism. After job1's execution, job1's current priority will be changed back to default priority, which is 1. Then job2 is executed, job1 is waiting. This is a loop.

Then job3 comes, job2, job1 will be waiting and their priority will be added, then job2 first has priority 2 and run. Then job2's priority go back to 1. Then job1 will have priority 2 and run, and priority go back to 0. So job3 is the next executed job. This is a loop.

- Understand the process of job scheduling: (Execution results and corresponding code)

Submit a new job：

Command: ./enq <jobname> <paras>

enq.c : Read the job_execution root and parameters, store it in ***struct enqcmd*** then store in FIFO.

```
write(fd,&enqcmd,DATALEN)< 0
```

scheduler.c: schedule() : read the cmd info from FIFO and store it to cmd

```
if ((count = read(fifo, &cmd, DATALEN)) < 0)
```

, run **do_enq() then schedule and switch**

Do_enq() fork a new process and in the child process ***raise(SIGSTOP)*** to stop this process. Listen for SIGCONT to continue.

Execute:

```
ce/OS/lab4  ./stat
/OS/lab4  ./enq -p 2 job4 2
/OS/lab4  ./stat
```

```
OK! Scheduler is starting now!!
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME               STATE   JOBNAME         CURPRI  DEFPRI


new job: jid=1, pid=25488
begin start new job
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME               STATE   JOBNAME         CURPRI  DEFPRI
1       25488   1000    10      1       Tue Mar 26 22:57:57 2019        RUNNING job4            3       2

normal termation, exit status = 0       jid = 1, pid = 25488
```

End of job execution:

The scheduler kill(child_pid,SIGCONT) to continue the child , child use execv to run the job and exit().

The signalhandler receive the signal SIGCHLD, free the memory and print message.

```
case SIGCHLD:
  ret = waitpid(-1, &status, WNOHANG);
  if (ret == 0 || ret == -1)
    return;

  if (WIFEXITED(status)) {
  #ifdef DEBUG
  printf("%d %d %d\n", ret, info->si_pid, current->job->pid);
  do_stat();
  #endif
    current->job->state = DONE;
    printf("normal termation, exit status = %d\tjid = %d, pid = %d\n\n",
      WEXITSTATUS(status), current->job->jid, current->job->pid);

  }  else if (WIFSIGNALED(status)) {
      printf("abnormal termation, signal number = %d\tjid = %d, pid = %d\n\n",
      WTERMSIG(status), current->job->jid, current->job->pid);

  } else if (WIFSTOPPED(status)) {
      printf("child stopped, signal number = %d\tjid = %d, pid = %d\n\n",
      WSTOPSIG(status), current->job->jid, current->job->pid);
  }
  return;
```

Execute:

```
OK! Scheduler is starting now!!

new job: jid=1, pid=22615
begin start new job
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME               STATE   JOBNAME         CURR
1       22615   1000    38      1       Tue Mar 26 22:39:40 2019        RUNNING job4            3

terminate job: 1
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME               STATE   JOBNAME         CURR
```

```
luning@luning-laptop ~/workspace/OS/lab4 ./stat
luning@luning-laptop ~/workspace/OS/lab4 ./enq -p 2 job4 100
luning@luning-laptop ~/workspace/OS/lab4 ./stat
luning@luning-laptop ~/workspace/OS/lab4 ./deq 1
id 1
luning@luning-laptop ~/workspace/OS/lab4 ./stat
```

Job scheduling due to Priority:

I modify the **select next()** to satisfy the requirement. Because in the original code, job is added at the end of the wait queue, so it is automatically a Round Robin under my implementation.

```
if (head) {
  for (prev = head, p = head; p != NULL; prev = p, p = p->next) {

    if (p->job->curpri > highest_pri) {
      select = p;
      selectprev = prev;
      highest_pri = p->job->curpri;
      max_wait_time = p->job->wait_time;
    }
    else if(p->job->curpri == highest_pri && p->job->wait_time >= max_wait_time)
    {
      select = p;
      selectprev = prev;
      max_wait_time = p->job->wait_time;
    }
    waitqueue_length++;
  }
```

Job scheduling due to time slice:

After a time slice, the process send a **SIGVTALRM** to himself, then call the **schedule** method to choose next job and switch.

```
switch (sig) {
case SIGVTALRM:
  schedule();
  return;
```

Conclusion:

This lab helps understand the scheduler. And I learned a lot about C language.

Submission(directly compress the following files, no more directory):

-lab4_report_studentID.pdf (pdf version report)

-scheduler.c (modified code .c file)