操作系统研讨课

蒋德钧

Fall Term 2018-2019

email: jiangdejun@ict.ac.cn
office phone: 62601007

中国科学院大学
University of Chinese Academy of Sciences

Lecture 3 Interactive OS and Process Management

2018.10.31

中国科学院大学
University of Chinese Academy of Sciences

Schedule

- Project 2 due
- Project 3 assignment

Project 3 Interactive OS and Process Management

- Requirement
  - Support interactive operation and basic process management
    - Implement a simple terminal and basic user commands
    - Implement four system calls: spawn, kill, wait, and exit
    - Implement three synchronization primitives
    - Implement inter-process communication mechanism: mailbox

Project 3 Interactive OS and Process Management

- Simple terminal
  - Screen
    - Use the provided printf to show input command
  - Shell
    - A user level process
    - Parse input command and invoke corresponding syscalls
    - Show the input command

Project 3 Interactive OS and Process Management

- Simple terminal

```
> [TASK] I am task with pid 2, I have acquired two mutex lock. (26)
> [TASK] I want to acquire a mute lock from task(pid=2).
> [TASK] I want to wait task (pid=2) to exit.









------------------- COMMAND -------------------
> root@UCAS_OS: ps
[PROCESS TABLE]
[0] PID : 0  STATUS : RUNNING
[1] PID : 1  STATUS : RUNNING
> root@UCAS_OS: spawn
> root@UCAS_OS:
```

Project 3 Interactive OS and Process Management

- Simple terminal
  - Note that shell runs after the kernel starts and acts as PID 1
  - In this project, shell polls the serial port instead of using interrupt
  - Please read the start code we provide to you

Project 3 Interactive OS and Process Management

- Process management
  - Spawn
    - starts a new process with a new process ID
    - corresponds to the program specified in the function’s arguments
  - Exit
    - Finish the running of a process in a normal way, and release all its resources

Project 3 Interactive OS and Process Management

- Process management
  - Wait
    - Waits on a process to complete its execution or to be killed
  - Kill
    - Sends signals to running processes to request the termination of the process, and release all its resources.

Project 3 Interactive OS and Process Management

- Implementing spawn
  - Shell command exec(id)
    - Given an array of tasks
    - Spawn the process with the number *id* corresponding to the array subscript
  - sys_spawn(task_t)
    - A syscall to start a new process
    - Fill the given *task_t* structure to initialize the PCB
    - Put the process into the ready queue

Project 3 Interactive OS and Process Management

- Implementing wait and exit
  - sys_waitpid(pid_t)
    - A syscall to wait on a process to terminate
    - Put the process into the corresponding wait queue
    - Note that, *pid* is hard coded in the task. If it is changed, pls. modify the task code
  - sys_exit(void)
    - Normally finish the running of the process
    - Reclaim all its resources

Project 3 Interactive OS and Process Management

- Implementing kill
  - Shell command kill(pid)
    - Kill the process with the corresponding *pid*
  - sys_kill(pid_t)
    - A syscall to kill a process immediately no matter which queue it is in
    - Reclaim resources, such as PCB, stacks
    - What else?

Project 3 Interactive OS and Process Management

- Implement basic process operations

```
> [TASK] I am task with pid 2, I have acquired two mutex lock. (144)
> [TASK] I want to acquire a mute lock from task(pid=2).
> [TASK] I want to wait task (pid=2) to exit.









------------------- COMMAND -------------------
> root@UCAS_OS: exec 0
exec process[0].
> root@UCAS_OS: exec 1
exec process[1].
> root@UCAS_OS: exec 2
exec process[2].
```

Project 3 Interactive OS and Process Management

- Synchronization – condition variable
  - Queue of tasks waiting on condition to be true
  - Monitor: condition variable + mutex lock
  - Main operations
    - Wait: block on a condition(if false) and release the mutex while waiting
    - Signal: unblock a waiting task once condition is true
    - Broadcast: notify all waiting tasks

Project 3 Interactive OS and Process Management

- Synchronization – semaphores
  - Control access to a shared resource
  - A value keeps track of the number of units of a resource that is currently available
  - Queues of waiting processes
  - Main operations
    - Down: decrement value and block the process if the decremented value is less than zero
    - Up: increment value and unblock one waiting process

Project 3 Interactive OS and Process Management

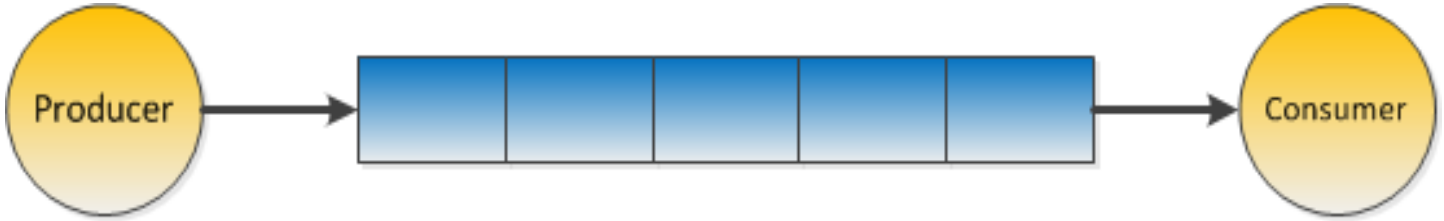
- Synchronization – barriers
  - A barrier for a group of tasks is a location in code where any task must stop at this point and cannot proceed until all other tasks reach this barrier
  - Keep track of the number of tasks at barrier
  - Maintain queue of waiting tasks
  - Main operations
    - Wait: block the task if not all the tasks have reached the barrier. Otherwise, unblock all

Project 3 Interactive OS and Process Management

- Synchronization
  - Note that
    - Pls. read the guide book and refer to the test case to see how these primitives are tested
    - These primitives are implemented in the kernel, and provide syscalls to user-level process
    - Pay attention to the impact of interrupt on implementing these primitives

Project 3 Interactive OS and Process Management

- IPC – Mailbox
  - Bounded buffer
    - Fixed size
    - FIFO
  - (Multiple) producers: put data into the buffer
  - (Multiple) consumers: remove data from the buffer



Project 3 Interactive OS and Process Management

- IPC – Mailbox
  - Producer-consumer problem
    - Two processes (producer and consumer) share a common fixed-size buffer used as a queue
    - The producer will not try to add data into the buffer if it is full
    - The consumer will not try to remove data from the buffer if it is empty

Project 3 Interactive OS and Process Management

- IPC – Mailbox
  - How to deal with producer-consumer problem?
    - Producer blocks if the buffer is full
    - Consumer blocks if the buffer is empty
    - How to notify the other part if the condition is satisfied?
      - Semaphore?
      - Condition variables?

Project 3 Interactive OS and Process Management

- IPC – Mailbox

Liu Bei

Sun Quan

Cao Cao

Project 3 Interactive OS and Process Management

- Step by step
  - Task 1
    - Implement shell process to support user command *ps* and *clear*
    - At least, *ps* shows two process (PID 0 and 1)
  - Task 2
    - Implement user command *exec* to invokde sys_spawn to run new process
    - Implement sys_exit and sys_wait
    - Implement user command *kill* to terminate a running process

Project 4 Synchronization Primitives and IPC

- Step by step
  - Task 3
    - Implement three primitives and verify them use the test cases
  - Task 4
    - Implement mailbox to test IPC and all your previous implementations, including synchronization, kill, wait, and spawn

Project 3 Interactive OS and Process Management

- Requirements for design review (40 points)
  - Which commands can be supported or will be supported by your shell?
  - What to do for spawn, kill, wait, and exit?
    - How do you handle synchronization when executing kill?
    - How about tasks in sleeping/blocking status when dealing with kill?

Project 3 Interactive OS and Process Management

- Requirements for design review (40 points)
  - How do you handle CV, semaphores, and barrier? What to do if timer interrupt occurs?
  - What is the structure for mailbox? How do you implement the mailbox?

Project 3 Interactive OS and Process Management

- Requirements of developing (60 points)
  - Implement shell and corresponding user commands (10)
  - Implement spawn (5), kill (5), wait (5), exit(5)
  - Implement the three synchronization primitives (15)
    - Condition variables, Semaphores, Barrier
  - Implement mailbox (15)

Project 3 Interactive OS and Process Management

- P3 schedule
  - P3 design review: 7th Nov.
  - Expected P3 due: 14th Nov.
- You are encouraged to enrich your own shell process to handle more user commands