操作系统研讨课
Course: B0911011Y

陈明宇、蒋德钧

Fall Term 2021-2022

email: cmy@ict.ac.cn / 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

2021.10.18

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

Schedule

- Project 3 assignment
- Project 2 due

Project 3 Interactive OS and Process Management

- Requirement I
  - Support interactive operation
    - Implement a simple shell terminal and support receiving and showing user commands
    - Implement a few shell commands, e.g. ps, clear

Project 3 Interactive OS and Process Management

- Requirement II
  - Support basic process management
    - Implement four system calls and the related shell commands
      - sys_spawn and its shell command: exec
      - sys_kill and its shell command: kill
      - sys_wait
      - sys_exit
    - Implement synchronization primitives: semaphores and barrier

Project 3 Interactive OS and Process Management

- Requirement III
  - Support inter-process communication
    - Implement synchronization primitives
      - semaphore
      - barrier
    - Implement inter-process communication mechanism
      - mailbox

Project 3 Interactive OS and Process Management

- Requirement IV
  - Support running processes on dual cores
    - Run dual cores
    - Support synchronization primitives、mailbox、binding core under dual cores

Project 3 Interactive OS and Process Management

- Simple terminal
  - Screen
    - Use the provided printf to show input command
  - Shell
    - A user level process: test_shell.c
    - 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 immediately 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 (test_shell.c) we provide to you

Project 3 Interactive OS and Process Management

- Basic process management
  - Spawn
    - starts a new process with a new process ID
    - Run the program specified in sys_spawn’s arguments
  - Exit
    - Finish the running of a process in a normal way, and release all its resources (e.g. lock, waiting processes)

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 all its resources (e.g. lock, waiting processes)

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_info_t *info, void* arg, spawn_mode_t mode)
    - A syscall to start a new process
    - Initialize the PCB and 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_t* 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, and lock

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

- You are encouraged to enrich your own shell to handle more user commands

Project 3 Interactive OS and Process Management

- Synchronization – semaphore
  - Control access to a shared resource
  - A value keeps track of the number of resources that are currently available
  - A queue holding waiting tasks
  - 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

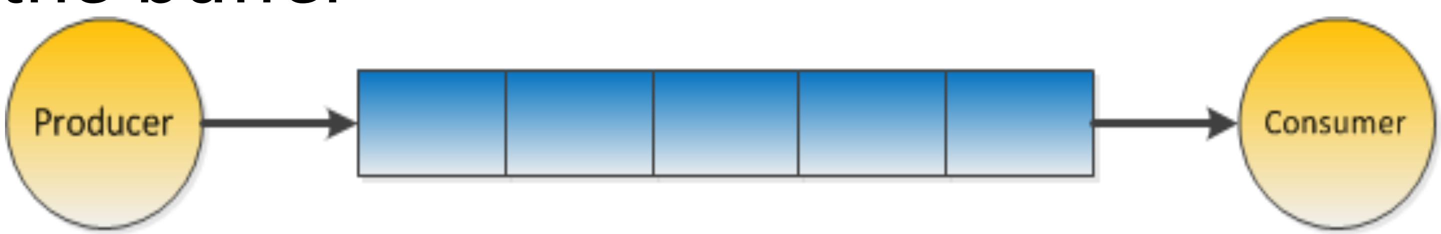
- Synchronization – barriers
  - A barrier for a group of tasks is a location in code where any task must stop at this point and cannot proceed until all other tasks reach this barrier
  - Keep track of the number of tasks at barrier
  - Maintain queue holding waiting tasks
  - Main operations
    - 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?
      - Use your implemented synchronization primitives

Project 3 Interactive OS and Process Management

- Running processes on dual cores
  - The two cores share the same memory
  - Each core has its own set of registers
  - How to start the second core?
    - By default, both cores start to run initially
    - The second core continues to loop with bbl

Project 3 Interactive OS and Process Management

- Start dual cores
  - After setup, at bbl use loadbootm instruction to start both cores
    - a0 register or mhartid register holds core id
    - You can use core 0 as the main core
    - Use sbi_send_ipi(const unsigned long *hart_mask) to send inter-process interrupt to the second core
    - Use the mhartid register value to decide which code path to run in your kernel

Project 3 Interactive OS and Process Management

- Access shared variables in kernel under dual cores
  - Shared variables in kernel
    - e.g. ready queue, variables in synchronization primitives
  - Use a big lock to protect the whole kernel when entering the kernel
    - Use atomic operations to implement lock
    - Pls. refer to arch/riscv/include/atomic.h
  - You need to implement some kernel variables separately for each core, e.g. current_running

Project 3 Interactive OS and Process Management

- Test dual-core
  - Accelerating adding operations with dual cores
    - Run adding with single core
    - Run adding with dual cores
    - Acceleration is expected when running dual cores
  - Tips
    - Use a relatively large clock slice

Project 3 Interactive OS and Process Management

- Test dual-core
  - Support mailbox with dual core
    - Three processes can both send and receive mails from mailbox
    - A process blocks when it fails to send or receive mail
    - Pls. do not let deadlock occur

Project 3 Interactive OS and Process Management

- Test dual-core
  - Implement shell command: taskset
    - taskset cpumask threadID
      - taskset 1 0: run thread 0 on core 0
    - taskset –p cpumask threadID
      - taskset –p 1 4: run thread 4 on core 1
    - Please read test_affinity.c to know how to test
    - A process has 5 sub-tasks, each task by default run on the same core with the parent process

Project 3 Interactive OS and Process Management

- Step by step
  - Task 1 (S-Core)
    - Implement shell process to support user command *ps* and *clear*
    - At least, *ps* shows two process (PID 0 and 1)
    - 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 2
    - Implement two primitives: semaphore and barrier. Verify them use the test cases.
    - Implement mailbox
    - S-Core: only need to implement barrier
    - A-Core: implement both primitives as well as mailbox

Project 4 Synchronization Primitives and IPC

- Step by step
  - Task 3 (A-Core)
    - Run dual-cores with the adding test case
  - Task 4 (C-Core)
    - Run mailbox under dual core
  - Task 5 (C-Core)
    - Support using taskset to bind process on specific core

Project 3 Interactive OS and Process Management

- Requirements for design review
  - Which commands can be supported or will be supported by your shell?
  - Show example code about spawn, kill, wait, and exit?
  - How do you handle the case when killing a task meanwhile it holds a lock?

Project 3 Interactive OS and Process Management

- Requirements for design review
  - How do you handle semaphores, and barrier? What to do if timer interrupt occurs?
  - Show the structure for mailbox. How do you protect concurrent accessing for mailbox?
  - How do you enable two CPU cores?

Project 3 Interactive OS and Process Management

- Requirements for design review
  - Any question about C-Core

Project 3 Interactive OS and Process Management

- P3 schedule
  - 25th Oct.: No class
  - P3 design review: 1st Nov.
  - P3 due: 8th Nov.