Lecture 3 Interactive OS and Process Management

2025.11.12

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
    - For unknown commands, just show "Unknown command"

Project 3 Interactive OS and Process Management

- Requirement II
  - Support basic process management
    - Implement four system calls and the related shell commands
      - sys_exec and its shell command: exec
      - sys_kill and its shell command: kill
      - sys_waitpid
      - sys_exit

Project 3 Interactive OS and Process Management

- Requirement III
  - Support inter-process communication
    - Implement synchronization primitives
      - Condition variables
      - 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 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 want to wait task (pid=3) to exit.
> [TASK] I am task with pid 3, I have acquired two mutex locks. (4737)
> [TASK] I want to acquire mutex lock1 (handle=0).









------------------- COMMAND -------------------
> root@UCAS_OS: exec waitpid &
Info: execute waitpid successfully, pid = 2 ...
> root@UCAS_OS: ps
[Process Table]:
[0] PID : 1  STATUS : RUNNING
[1] PID : 2  STATUS : BLOCKED
[2] PID : 3  STATUS : READY
[3] PID : 4  STATUS : BLOCKED
> root@UCAS_OS: ss
Error: Unknown Command ss!
> 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

Project 3 Interactive OS and Process Management

- Basic process management
  - Exec
    - Starts a new process with a new process ID
    - Run the program specified in the arguments of sys_exec
  - 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
    - Wait on a process to complete its execution or to be killed
  - Kill
    - Send signals to running processes to request the termination of the process, and release all its resources (e.g. lock, waiting processes)

Project 3 Interactive OS and Process Management

- Implement exec
  - Shell command: exec id/name
    - Spawn the process with the number id or the program name
    - sys_exec syscall to start a new process
    - Initialize the PCB and put the process into the ready queue

```
pid_t sys_exec(int id, int argc, uint64_t arg0, uint64_t arg1, uint64_t arg2);

pid_t sys_exec(char *name, int argc, char **argv)
```

Project 3 Interactive OS and Process Management

- Implement wait and exit
  - sys_waitpid(pid_t)
    - A syscall to wait on a process to terminate
    - Put the process into the corresponding wait queue
  - sys_exit(void)
    - Normally finish the running of the process
    - Reclaim all its resources

Project 3 Interactive OS and Process Management

- Implement 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
  - By default, shell waits the termination of the current running process by executing sys_waitpid
  - Pls. add & to allow shell to continue execute without waiting for current process

```
-------------------- COMMAND -------------------
> root@UCAS_OS: exec waitpid &
Info: execute waitpid successfully, pid = 2 ...
> root@UCAS_OS: exec barrier &
Info: execute barrier successfully, pid = 5 ...
> root@UCAS_OS: exec condition
Info: execute condition successfully, pid = 9 ...
```

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 – 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

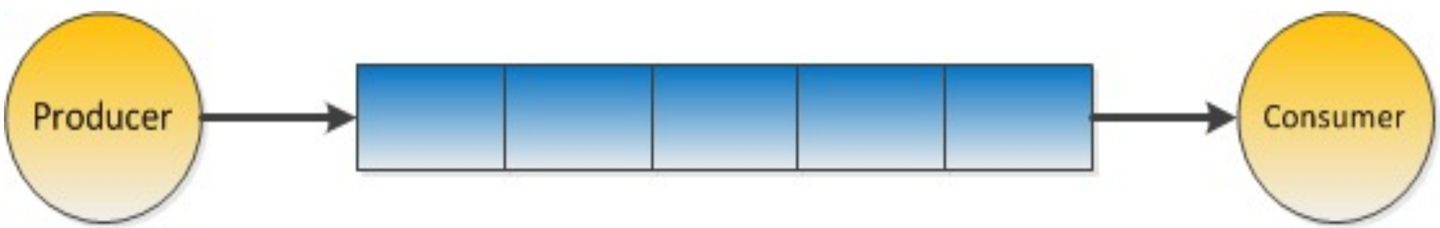
- Synchronization – condition variables
  - Queue of tasks waiting on condition to be true
  - Monitor: condition variable + mutex lock
  - Main operations
    - Wait: block on a condition(if false) and release the mutex while waiting
    - Signal: unblock a waiting task once condition is true
    -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

Project 3 Interactive OS and Process Management

- Synchronization
  - 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 and L1 cache
  - 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 power on, use loadbootm instruction to start both cores
    - *a0* register or *mhartid* CSR register holds core id
    - Use the *mhartid* CSR register value to decide which code path to run in your kernel
    - You can use core 0 as the main core
    - Use *send_ipi(const unsigned long *hart_ mask)* to send inter-process interrupt to the second core

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, ready queue

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

- taskset
  - Implement shell command: *taskset*
    - taskset cpumask taskname
      - taskset 1 mytask: run mytask on core 0
    - taskset –p cpumask pid
      - taskset –p 2 4: run thread 4 on core 1
    - Please read 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
    - Extend *ps* command to show the ID of the core where the current processes are running

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)
  - Implement user command *exec* to invokde sys_exec to run new process
  - Implement sys_exit and sys_wait
  - Implement user command *kill* to terminate a running process

Project 3 Interactive OS and Process Management

- Step by step – Task 2
  - Implement two primitives: barrier and condition variables
  - Verify them using the test cases
  - Implement mailbox

Project 3 Interactive OS and Process Management

- Step by step – Task 3
  - Run dual-cores with the test case
- Step by step – Task 4
  - Support using taskset to bind process on specific core

Project 3 Interactive OS and Process Management

- Step by step – Task 5
  - Processes A and B concurrently access mbox1 and mbox2
    - A first reads from mbox2, and then writes to mbox1
    - B first reads from mbox1, and then writes to mbox2
    - At certain time, mbox1 and mbox2 are fulfilled, meanwhile both A and B try to write mbox1 and mbox2, then deadlock occurs
  - Re-implement the above deadlock scenario for mailbox accessing

Project 3 Interactive OS and Process Management

- Step by step – Task 5
  - Design and implement threads for mailbox accessing to avoid deadlock
    - E.g. for Process A, one thread is responsible for reading from mbox2, and the other is for writing to mbox1

Project 3 Interactive OS and Process Management

- Requirements for design review
  - 请展示spawn、kill、wait和exit的伪代码
  - 当kill一个持有锁的进程时，kill的实现中要进行哪些处理？
  - 如何实现条件变量、屏障？请简述你的设计思路或展示伪代码。在使用条件变量、屏障时，如果有定时器中断发生，你的内核会做什么处理么？

Project 3 Interactive OS and Process Management

- Requirements for design review
  - 简述如何保护mailbox并发访问的正确性？
  - 简述如何让主从核分别正常工作？
  - 关于C-Core的任何问题

Project 3 Interactive OS and Process Management

- Requirement for S/A/C-Core

Core type	Task requirements
S-Core	Task 1 and Task 2 only needing to implement barrier
A-Core	Tasks 1, 2 and 3
C-Core	Tasks 1~5 and all test cases run under dual cores

Project 3 Interactive OS and Process Management

- P3 schedule
  - P3 design review: 19th Nov.
  - P3 due: 26th Nov.