操作系统研讨课

蒋德钧

Fall Term 2017-2018

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

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

Lecture 4 Synchronization Primitives and IPC

2017.11.15

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

Schedule

- Project 3 due
- Project 4 assignment

Project 3 Due

- P3 due
  - We test clock interrupt handler and blocking sleep, as well as priority-based scheduler
  - Please compile your code, running the code on your board, and show the results to TA
  - Answer any questions we may ask

Project 4 Synchronization Primitives and IPC

- Requirement
  - – Support synchronization primitives, process management, and inter-process communication
    - Implement three system calls: spawn, kill, and wait
    - Implement three synchronization primitives
    - Implement inter-process communication mechanism: mailbox

Project 4 Synchronization Primitives and IPC

- Operations on processes
  - Fork
    - creates a copy of the parent process
    - usually then calls exec to start a new program by overlaying itself with the new program
  - Spawn
    - starts a new process with a new process ID
    - corresponds to the program specified in the function’ s arguments

Project 4 Synchronization Primitives and IPC

- Operations on processes
  - 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

Project 4 Synchronization Primitives and IPC

- Implementing spawn
  - do_spawn(char *filename)
    - A syscall to start a new process
  - How to initialize a process?
    - PID
    - Entry point
    - PCB initialization

Project 4 Synchronization Primitives and IPC

- Implementing spawn
  - do_spawn(char *filename)
    - A syscall to start a new process
  - Note that
    - Entry point: we provide ramdisk emulation, pls. refer to files.c in the test case and ramdisk.c
    - PCB initialization: we provide an array of PCBs with the size NUM_PCBS for reservation, use the available PCB slot in this array

Project 4 Synchronization Primitives and IPC

- Implementing wait
  - do_wait()
    - A syscall to wait on a process to terminate
  - What to do for do_wait()?
    - Possible solution: put the process into the other’ s wait queue

Project 4 Synchronization Primitives and IPC

- Implementing kill
  - do_kill()
    - A syscall to kill a process immediately no matter which queue it is in
  - What to do for do_kill()
    - Reclaim resources, such as PCB, stacks
    - What else?

Project 4 Synchronization Primitives and IPC

- 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 once condition is true
    - Broadcast: notify all waiting tasks

Project 4 Synchronization Primitives and IPC

- 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 4 Synchronization Primitives and IPC

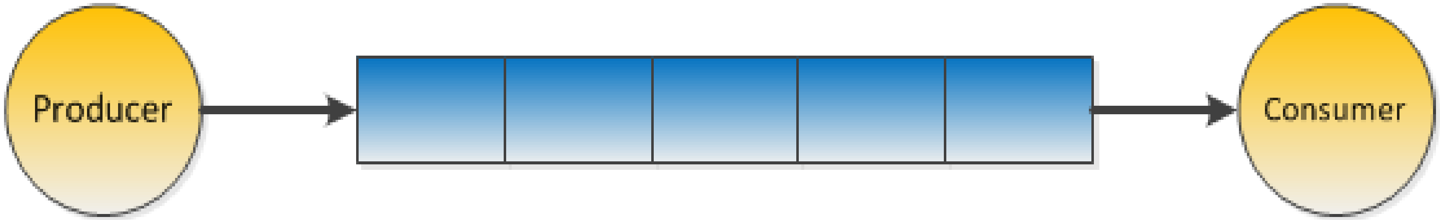
- Synchronization – barriers
  - A barrier for a group of tasks is a location in code where any task must stop at this point and cannot proceed until all other tasks reach this barrier
  - Keep track of the number of tasks at barrier
  - Maintain queue of waiting tasks
  - Main operations
    - Wait: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

Project 4 Synchronization Primitives and IPC

- Synchronization
  - Note that
    - Pls. refer to the test case to see how these primitives are used
    - Pay attention to the impact of interrupt on implementing these primitives

Project 4 Synchronization Primitives and IPC

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



Project 4 Synchronization Primitives and IPC

- 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 4 Synchronization Primitives and IPC

- 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 4 Synchronization Primitives and IPC

- Step by step
  - Task 1
    - Implement do_spawn to allow your code to run new process (test_spawn)
    - Implement do_kill and do_wait, but we test it until task 3
  - Task 2
    - Implement three primitives and verify them use the test cases (test_barrier, test_all)

Project 4 Synchronization Primitives and IPC

- Step by step
  - Task 3
    - Implement mailbox to test IPC as well as do_kill and do_wait (test_sanguo)

Project 4 Synchronization Primitives and IPC

- Bonus （2 points）
  - How to deal with locks when a task is killed?
    - Need to implement syscall for lock_acquire
    - Implement your own test cases, e.g. two user-space processes acquiring locks and then one process is killed

Project 4 Synchronization Primitives and IPC

- Requirements for design review (40 points)
  - What to do for spawn, kill and wait?
    - How do you handle synchronization primitives when dealing with kill?
    - How about tasks in sleeping/blocking status when dealing with kill?
  - How do you handle CV, semaphores, and barrier? What to do if timer interrupt occurs?

Project 4 Synchronization Primitives and IPC

- Requirements for design review (40 points)
  - What is the structure for mailbox? How do you deal with the general producer-consumer problem?

Project 4 Synchronization Primitives and IPC

- Requirements of developing (60 points)
  - Implement do_spawn (5)
  - Implement do_kill (5)
  - Implement do_wait (5)
  - Implement the three synchronization primitives (20)
    - Condition variables, Semaphores, Barrier
  - Implement mailbox (25)

Project 4 Synchronization Primitives and IPC

- P4 schedule
  - P4 design review: 22nd Nov.
  - P4 due: 29th Nov.