Syncronization primitives-IPC

November 19, 2017

All the operation should only be performed in kernel state

1 spawn, kill and wait

1. spawn:
   (a) enter_critical, process spawning should not be interrupted.
   (b) find an unused slot in the pcb table, take the index plus one as pid
   (c) find target process information in files table in files.c
   (d) initialize pcb of this process
   (e) in files.c, we may find information for entry point, task type. store them in a task_info structure and pass it to initialize_pcb
   (f) leave_critical. Scheduling is not necessary
2. do_wait:
   (a) enter_critical
   (b) mark target process's status as blocked(proper?).
   we may need to add a wait_queue field to pcb structure. Each process should maintain a wait queue. Processes in this queue are the ones waiting for this process to terminate. Once this process is killed or terminates, release all the process from this queue. So function exit should be changed.
   (c) put this process to a wait queue
3. do_kill:
   (a) enter_critical
   (b) mark target process's status as EXITED so that the slot of this process in pcb can be used by other processes.
   (c) release its stack:(seems there is no efficient way to restore stack space)
   (d) release all the locks it holds.(how to find all the lock a process has held)
   (e) leave_critical

2 Syncronization primitives

all the operation except init should not interrupted.

1. Condition variables
   structure (a) only a blocked queue is needed. Maybe some debug information could be added.
   operation all the operation should not be interrupted
   (a) init: initialize the wait queue in the condition vairable
   (b) wait: wait for a condition variable, relaese the lock the task holds. When a task is finalliy unblock, reacquire the lock held before.
   (c) signal: unblock only one process from condition variable's wait queue

(d) broadcast: unblock all the processes from condition variable's wait queue

2. semaphore

structure (a) an interger as semaphore counter

(b) a blocked queue is needed as well

operation (a) init: initialize a semaphore with given value, which should be greater or equal to zero and initialize its blocked queue

(b) up: increase semaphore value by one, if the value before increment is zero, unblock a task from blocked queue.

(c) down: decrease semaphore value by ones if the value before decrement is not already zero, or block current process.

3. barrier

structure: (a) an integer *target* indicating number of total tasks to wait

(b) an integer *blocked* counting tasks already blocked

(c) a blocked queue is need as well

operation: (a) init: initialize *target* with an integer, which should be positive

(b) wait: if *blocked* equals to *target-1*, don not block current task, unblock all the tasks in blocked queue and set *blocked* to 0, or block current task.

3 mailbox

structure: 1. name: the name of this mail box

2. users: record the number of tasks using this mailbox. If it is zero, restore this mail box

3. used: whether this mail box is used or not.