Homework 2

ComS 352 | Gavin Monroe | gmonroe

|  |
| --- |
| 1. **Ex 3.8 Describe the actions taken by a kernel to context-switch between processes.** |
| There are a couple of reasons, but shortly they are as follows:   * In reaction to a clock interrupt, the OS saves the PC and consumer stack pointer of the currently executing process, and transfers control to the kernel clock interrupt handler. * Clock interrupt handler saves the registers, machine state, such that the state of the floating point registers, in the process PCB. * Operating System invokes the schedule to determine the next process to execute. * In response to a clock interrupt, the OS saves the PC and patron stack pointer of the currently executing process, and transfers manipulate to the kernel clock interrupt handler. |
| 1. **Ex 3.11. Including the initial parent process, how many processes are created by the program shown in Figure 3.32? Draw a process tree to show the relationship between the processes.**  |  |  | | --- | --- | |  | **Answer: Program loops 4 times. (2^n -1)** | |
| **3. Ex 3.12 Explain the circumstances when the line of code marked printf("LINE J") in Figure 3.33 is reached.** |
| This line will execute in the child process. That being said the pid must be 0, because of the else if statement. When we are in the child process, the system call “execlp” executes the command “ls” which will overall replace the child process with a brand new “ls program” file and a new program starts execution in its main function. That being stated the pid of the process will not change because “exec” command is replacing the image of the child process, not creating a new one. |
| **4. Ex 3.13 Using the program in Figure 3.34, identify the values of pid at lines A, B, C,and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)** |
| A: 0, B: 2603, C: 2603, D: 2600.  Parent: pid = 2603 for C and for the parent PID for D is 2600, for the child pid for A its 0, and for the child pid for B its 2603. Fork will return the process id which is created by the fork in the parent process. |
| **5. Ex 3.16 Using the program shown in Figure 3.35, explain what the output will be at lines X and Y.** |
| As the discern data is simply copied to the infant (no longer reference) , modifications made in child does not replicate over determined.  Hence the values of the Child at the road X are 0,-1,-4,-9,-16. Whereas the values of discern continue to be unchanged.  The values of the discern at line Y are 0, 1, 2, 3, 4 |
| **6. In a multitasking system, the context-switch time is 1ms and the time slice is 10ms. Will the CPU efficiency increase or decrease when we increase the time slice to 15ms? Explain why.** |
| It relies upon on the advent time and burst time of the processes. On increasing the time slice the waiting and flip around time can growth and reduce both. As ready time and turn around time are the fundamental standards for calculating the performance, so we can't say whether or not the performance will boom or decrease. So the answer is It relies upon on the advent time and burst time of the processes. |
| **7. Give an example event that will cause the following process state transition:**  **(a) from running to waiting;**  **(b) from running to ready;**  **(c) from waiting to ready** |
| **For (a): From running to waiting: (running) ---->> (waiting).**  Here transition occurs because process needs input or output operation then the process temp removes the running state and enters into waiting state. Example would be if p1 process needs 3 second cpu time, 10 second input, 3 second cpu time. P1 enters into running state after 3 seconds then enter into waiting state for 10 second input or output operations then enter into running state for 3 second cpu time.   |  |  |  | | --- | --- | --- | | **RUNNING state** | 10 second I/P →  ← 3 second CPU time | **WAITING State** |   **For (b): from (running) → (ready)**  This transition occurs sometimes the process in running state have less priority when another process in ready state has high priority then high priority process enters into running state and low priority process enters into ready state.  P! = priority high, P2 = priority low   |  |  |  | | --- | --- | --- | | **RUNNING state** | ← interruption  ← P2  P1 → | **READY State** |   **For (c): from (waiting) → (ready)**  After completion of input or output operations in waiting state then enters inot ready state for scheduled dispatch. **(ready) <<--- (waiting)** (I/P complection)  Just like (a) Process P1 enters into ready state after complete 10sec of input or output operations in waiting state. |