CSE320

Take-Home Quiz 2

Name: Umme Abira Azmary

ID : 20101539

Section : 06

Ans. to the ques. no-01:

The message passing system ~~can~~ can be used to achieve inter-process communication in this scenario.

This system allows processes to communicate and to synchronize actions through a distributed environment where the communicating processes may reside on different computers connected by a network. However, it is not suitable for exchanging large amounts of data as this system communicates using a queue where messages of all the available processes are shared, so if the messages or sharing data amounts are large, the queue of message passing system will overload. As a result, exchanging small amounts of data is preferable.

Ans. to the ques. no-02:

The wait() system call need to be used in this scenario as it will allow the ~~parent~~ child process to complete its execution first.

This system call ensures the ~~ex~~ complete execution of the child process by ~~switching~~ pausing the parent's execution ~~to chil~~ and the parent process can only resume its execution when it can retrieve the

exit status of the child process. In absence of this system
a parent process ~~will~~ continues its execution along with
the child process and sometimes parent processes
execution may be finished before the child process whic
can result in execution of orphan process.

Ans. to the ques. no-03:

There can be multiple reasons for which the child
process starts ~~execution~~ execution from the next instruction after the
fork:

(i) If a child process starts execution from the same
line that has created it, it can ~~at~~ again create
another child and so on. This means, the process ~~can~~
will be creating child one after another ~~wt~~ without
actually completing its execution.

(ii) Moreover, the different PID value of fork helps to ~~ge~~
satisfy different instructions of a same code by the

parent and child process. To explain, in the parent process, the fork() returns the PID of child, ~~to~~ ~~[illegible]~~ and 0 as PID to child itself. As a result, running from the next Line ensures the execution of different logic than the parent's or it can faster the ~~proc~~ execution of parent's process.

Ans. to the ques. no: 04 :

A process from its creation till its completion is handled by different schedulers based on different performance decisions. The different states working with different schedulers are:

New: The long term scheduler passes a program that is created as a new process from job queue to ready queue based on its performing time while ensuring the utilization of maximum performance of CPU.

Ready: The short term scheduler passes a ready process from Ready queue to CPU very frequently ensuring the CPU must not be idle for a moment.

Running: The Medium Term Scheduler manages the swappi
of processes in CPU to optimize memory usage.

Waiting: The Medium Term Scheduler swaps out a process out of the CPU when an I/O operation invokes or excedes the assigned time ~~a~~ so that it can go back to ready state after performing necessary execution.

Terminated: The short term scheduler helps CPU to move ~~clean~~ out a finished process to ensure no ~~space~~ resources are wasted by a terminated or ~~or~~ idle process.

Ans. to the ques. no-05:

If n processes to be scheduled on one processor, when the processes can't be interrupted, the number of different schedu--les ~~are~~ possible to be executed are n!.

$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots\ldots \times 1.$

Here, the first process is being chosen from n ~~et~~ numbers of options. then, the second process is being chosen from (n-1) numbers of options are so on. The important point is, while running a process, it can't be interrupted ~~as so~~ and consequently, n! numbers of different schedules are possible to run n numbers of processes.

Ans. to the ques. no-06:

To ~~consider~~ maximize the amount of CPU time allocated to the user's process while implementing a multilevel queue scheduling, the user's processes can be stored in Foreground queue and this foreground queue can have 2/3 times of the CPU times while 1/3 time can be used to process the background queue.

Moreover, different algorithms can be used in foreground to ensure smooth transictions of processes. For example: Round Robin and premptive priority can be applied on foreground in seperated space so that these can transit the user's processes while providing better performance.

<u>Ans. to the ques. no- 7:</u>

To ~~run~~ implement SRTF In handling dynamic workloads throughout a day, potential challenges that can be encounted are:

(i) SRTF does not account for priority, as a result, ~~high~~ long running processes with high priority can be delayed in favor of lower-priority, small processes. For example: a coding simulation may require more processing time because of complexity than random browsing, however, running coding faster may be needed and it won't be addressed by using SRTF algorithm.

(ii) With the arrival of multiple processes simultaneously, the ~~process~~ CPU needs to calculate and switch to Short brust processes. This heavy context switching throughout a whole day may result as a delay in CPU performance as CPU will face idle times with every switching.