IMPLEMENTATION DETAILS - AYUSH JAIN [jain207@purdue.edu]

Specific questions asked in lab3 for implementation are at the end

Steps taken for the implementation

1. The *mysuspend* system call on a particular process uses *delayed_suspend_flag* in the process table entry (*struct procent*) to indicate that the process should be suspend when it becomes ready when the process is currently in a waiting state such as PR_WAIT, PR_RECV, PR_SLEEP, or PR_RECVTIM.
2. For a process that has a state of PR_CUR, PR_READY or PR_SUSP, this system call behaves exactly the same as the suspend system call
3. The system call returns an error for invalid process id or a null process.
4. In order to handle the delayed suspend flag, changes are made in ready call as all system calls like signal, send, resume etc. calls ready system call to change the process's state to PR_READY and adding the process to the ready list.
5. If the flag is set, then it is time for the process to be suspended. Thus, the process's state is changed to PR_SUSP and flag is again set to false for future *mysuspend* system calls.

Answers to specific questions asked :

Although deferred suspension is straightforward, the semantics of deferred actions can become troublesome. For example, suppose a process is current waiting, but has the deferred suspension flag set, and the process is resumed.

a) Should the resume cancel the deferred suspension, or should the process first be moved to the suspended state and later resumed?

Yes, the resume system call should cancel the deferred suspension as done in the implementation in *resume.c.* The idea behind the delayed suspension is to suspend the process till it is eligible to resume. If resume system call is invoked while the process is in waiting state (PR_WAIT, PR_RECV, PR_SLEEP, or PR_RECVTIM) , then it means that process can resume as soon as it's current waiting ends.
It is an additional overhead to keep track of the resume calls while process is in waiting state and there is no use sending process to suspended state after it's waiting state ends.

b) What should we do if some action is valid when a process is in the suspended state, should the the action happen when the process has deferred suspension?

The action shouldn't happen when the process has deferred suspension as it may be dependent on the current waiting state of the process [deferred suspension case]. Allowing such an action to execute can result in error in order of overall computations of the program.

For Example : Resume is called on the suspended process on the basis of an event which has some dependency with the reason of waiting state of process (like semaphore on a buffer in case of wait() system call). In usual suspended case, the event requiring buffer can execute normally but not in case of delayed suspension case.
So, suspend system call should be called at the end of its waiting state and resume system call after the process is in suspend state.

c) Should the operating system keep a list of all the actions to be performed once the process is out the deferred situation?

Yes, the list of actions can stored in a queue as maintaining order is necessary in such a scenario as actions can be dependent on each other. Also, number of attempts can be recorded for each action as done in current implementation of XINU for deferred attempts for resched() system call.

The additional overhead in terms of space and time complexity will allow complete and correct order of execution of all actions.

[EXTRA CREDENTIAL]

1. Mysleepms system call has been implemented using my_delay integer value in the struct procent for each process.
2. If the process identifier given is the currently executing process, then mysleepms works exactly the same as sleepms.
3. If the process identifier given is not the currently executing process then, the process with process identifier given is put to sleep for the specified number of milliseconds as soon as it becomes ready. This is handled by recording the delay required in the my_delay integer for each process table entry.
4. This delay value is then used when ready system call is invoked (directly/indirectly) to change the process's waiting state.
5. Thus, as soon as the process becomes eligible to run, the process is put to sleep for the specified number of seconds.

Handling of specific situations as mentioned in the assignment:

1) Process 'A' is not eligible to run and another process calls mysleems, using the process identifier for 'A' and a delay greater than zero, multiple times before 'A' becomes eligible to run.

The first mysleep system will be successful in this case storing the delay value which will bw used as soon as process is out of waiting state. Successive mysleep calls after that will result in error.

2) Process 'A' is currently sleeping and another process calls mysleepms using the process identifier for 'A' and a delay of 0 milliseconds.

If the process is currently in sleeping state, any successive calls to mysleep in that state will result in error.

3) Process 'A' is not eligible to run, but mysleepms was called causing it to sleep for a time greater than 0 when it becomes ready. Before it gets a chance to sleep, another process calls mysleepms using the process identifier for 'A' and a delay of 0 milliseconds.

The first mysleep system call will be successful and successive mysleep system calls to the process will result in error irrespective of delay > 0 or delay =0.

---

a) Also, in case delay = 0 and process is not the current process, the mysleepms simply returns from the system call without any use of yield as yield system call is used in sleepms() as it is in the context of currently executing process.

b) Similar checks of bad pid, null process and range of delay between 0 and MAXSECONDS are performed for the mysleepms system call.

c) In case the process is in ready state, it is changed to sleep state with a specific amount of delay as required.