Aaron Althoff

CS354

# Lab 6 Answers

3. The main modification I made to XINU was to the process table entry structure, to which I added multiple fields. I added fields prfirstsig, prsecondsig and prthirdsig to handle each of the three signals and maintain the order they were sent in (filling prfirstsig first, then prsecondsig and prthirdsig) and then when running the handlers I could handle the signal in the first then the second and then the third. By not removing the signal from the field in the process table until after the callback function has completed execution, it fixes the issue raised about back-to-back invocation, because I do not allow a signal to be added to any of the fields if it already exists in one of them. As a result if the callback is executing then the signal is still occupying one of the fields and it cannot be added until after callback execution has finished to prevent data structures from being corrupted. I also added the field prchldterm which represents the pid of the first child process to terminate and is used by childwait to return the correct pid. Prchlnum is used to keep track of how many child processes a process has, and if it is 0 childwait returns SYSERR. To handle the XSIGCHL signal I modified the kill function to send the signal to the parent process when the child finishes. It also sets the parent process to ready if it is blocking with state PR_CHLDWAIT. To implement the XSIGXTM signal I put a condition in the clkhandler() function to check if a process has gone over its allotted time and if it has it sends a signal to the process. ***I WAS NOT SURE IF THE SIGNAL SHOULD BE SENT MORE THAN ONCE TO A PROCESS, SO ONCE THE SIGNAL IS HANDLED BY THE PROCESS, IT IS SENT AGAIN. TO FIX THIS I WOULD HAVE ADDED A BOOLEAN FIELD TO THE PROCESS TABLE ENTRY WHICH WOULD BE SET TO TRUE THE FIRST TIME THE XSIGXTM SIGNAL IS SENT TO MAKE SURE IT IS NOT SENT MORE THAN ONCE.*** To test this I used a function called testgc initially I tested the signals individually to make sure that they worked in every way. Then I tested them in different orders to ensure that they were handled in the correct order.

4. For the garbage collection my plan was to create a structure called memstruct that had a pointer to another memstruct to form a linked list, a char * to represent the beginning address of a memory block allocated by getmem and an int to represent the size of the memory block. I was going to create a new memblock each time memory is allocated (in getmem) and add it to the linked list stored in the process table entry for a process. In freemem I would remove the memstruct from the linked list corresponding to the memory block being freed. When a process was killed by the kill system call I would go through the linked list and free each memory block that was not explicitly freed by freemem. I was not able to get the struct to work in getmem so I was not able to actually implement garbage collection. To verify correctness I would have gone through the free list after a process is freed to ensure that all of the memory locations allocated were returned to the free list after it has terminated even if freemem was not called on it.