CS 354
Gray Houston
Dr. Park

Lab 01 Answers

4.) Static Priority Scheduling

1.)

Running the specified code when INNERB has a value of 1,000,000 produces the following output: “PAPABPABCPABCDABCDBCDCDD”. The same output occurs every single time the code is executed. The code for the printloop() function is within the printlooop.c file under the system/ directory.

The ‘P’ characters are interleaved by all of the main process’ children who are running. Because the main process and its children all possess the same scheduling priority and the fact that XINU uses static scheduling, all the main process and all of its children are executed in a circular fashion such that each process receives an equal amount of time running. Because each child process can only print a single character per context switch, a single character being printed to the screen represents that process being switched onto the CPU.

Now, let’s explain the output (“PAPABPABCPABCDABCDBCDCDD”) in terms of context switching. First, ‘P’ is printed (using the putchar() method) and then resume is called on the “A” child process. When resume is called, “main” yields its processing time and is switched out. At this point, each child process that has been resumed will execute sequentially. Because “A” is the only ready child process, ‘A’ is the only additional character printed before the “main” process is switched back onto the CPU. Once another ‘P’ is printed, the “B” child process is resumed and “main” yields its time. Then, because “A” and “B” are the only child processes in the ready state, ‘A’ and ‘B’ are printed sequentially to the screen. This pattern will continue similarly until the main process goes to sleep by calling sleepms(). Once this function is called, the “main” process moves out of the ready state and therefore only the child processes will be executing. The child processes execute sequentially until they finish and move out of the ready state.

2.)

In this case, this is the resulting output: “PAPABPABCPDDDDDABCABCBCC”. Notice in the middle of the output how there is a series of 5 ‘D’ characters in a row. This coincides with the “D” process being moved into the ready state by resume() being invoked. When the method is resumed, the “main” process yields its processing time and a context switch will occur. XINU will look at the table of all available processes to determine which process to switch onto the CPU. Because the “D” process will have a greater priority than any other process, it will be selected and ‘D’ will be printed to the screen. Once the “D” process has used all of the time in its quanta, the scheduler will look at the process table and choose the process with the largest priority. Because “D” still has the highest priority, it will execute again. Because the “D” process has the greatest priority, it will always execute before all other lower priority processes until “D” moves out of the ready state (in this case, when the process finishes executing). This explains why there is a series of 5 ‘D’ characters in the output

3.)
Here is the output: “PAPABPCCCCCABPDDDDDABABB”. The interesting segments of this output are the series of ‘C’ characters closely followed a series of ‘D’ characters. The reasons for this output are similar to the answer for part 2 of this question. Before the “C” process is resumed, the behavior is the same as in part 1, where all ready child processes and main process have the same priority and therefore execute sequentially (like in a round robin). Once the “C” process is resumed, it is the process with the greatest priority and will execute until it finishes, producing a series of 5 ‘C’ characters. Then, the round robin behavior will be resumed such that “A” executes followed by “B” and then “main”. At this point, an additional ‘A’, ‘B’, and ‘P’ characters have been printed to the screen. Then, the “D” process is resumed and the scheduler is invoked to determine which process should execute. Because “D” has the highest priority, it will execute without interruption until it finishes and print 5 ‘D’ characters. This behavior of processes “C” and “D” is due to the fact the XINU uses static scheduling and will always execute the process with the highest priority and never updates a process’ priority once it has been created (as opposed to dynamic priority scheduling).

5.) Stack Layout of Concurrent XINU Processes