<u>ECE 592 – Operating Systems Design</u>

<u>PROJECT: 1 REPORT</u>

Name: Saurabh Labde
Student ID:200206275

1) What is the ready list.? List the system calls that operate on it.

Ans: Ready list is a queue of all the processes which are in the Ready state and are available for scheduling at the next scheduling event. When a process goes in the ready state, it is placed in the ready list in decreasing order of priority, i.e. the process at the head of the queue has the highest priority and the the process at the tail of the queue has the lowest priority. The system calls operating on ready list are:

A) Resume
B) Suspend
C) Kill
D) Wait
E) Ready
F) Receive
G) Sleep
H) Wakeup
I) Clockintr *

2) What is the default process scheduling policy used by XINU? Can this policy lead to process starvation.?

Ans: - Xinu makes use of Priority based scheduling as its default policy. Yes, this policy can lead to process starvation. In priority-based scheduling, process with the highest priority is scheduled first. If multiple processes have the same priority, they are scheduled in a round robin manner. Thus, as long as there are high priority processes in the ready queue, they will keep scheduling until they move out of the ready state. Thus, the process at the bottom of the ready queue which has a lower priority will never be scheduled as long as higher priority processes are in the queue. This leads to the starvation of the lower priority processes.

3) When a context switch occurs, how does the resched function decide if the currently execution function should be put back in the ready list.

Ans: - The resched function is not directly called by the user but is called at the end of time slice or by system calls when a process changes state. Thus, in the resched function, the state of the process is checked. If the process is in the current state and has the highest priority in the ready list, then it is directly scheduled without being put in the ready list. If there is a process in the ready list which has a priority higher than the given process, then the given process is put in the ready list according to its priority. If the process was not in the current state, it is not inserted back into the ready list but has to wait till its state changes to ready.

4) Explain what is a stack frame and why is it required in function call.?

Ans: - The specific stack associated with each function is called a stack frame. During a function call, the current function is stopped, and the context is switched to the new function that is being called. But once we return from the function being called, we should see the state of the caller function as it was before the function call. Stack frames are used to store the state the of the function during a function call. It contains space for local variables, registers, return address and other items needed for the computation by the function.

5) The function ctxsw takes two parameters. Explain the use of these parameters.? Which assembly instruction sets the instruction pointer register to make it point to the code of the new process.

Ans – ctxsw receives two arguments, the location of the process table where the current process stack pointer should be stored and the address in the process table that contains the new stack pointer . Therefore, a single instruction stores the stack pointer at the location given by the first argument and another instruction loads the stack pointer from the address as indicated by the second argument field.

6) Analyze Xinu code and list all the circumstances that can trigger a scheduling event.

Ans: - A) Resched is called by the timer interrupt (clkhandler) at the end of every time slice.
B) When a process is killed, resched is called and scheduling takes place.
C) When a process enters the sleep state or is put to sleep, resched is called
D) When a process voluntarily yields the processor, resched is called.
E) When a process is resumed and inserted in the ready list, resched is called which triggers scheduling.

Coding assignments:

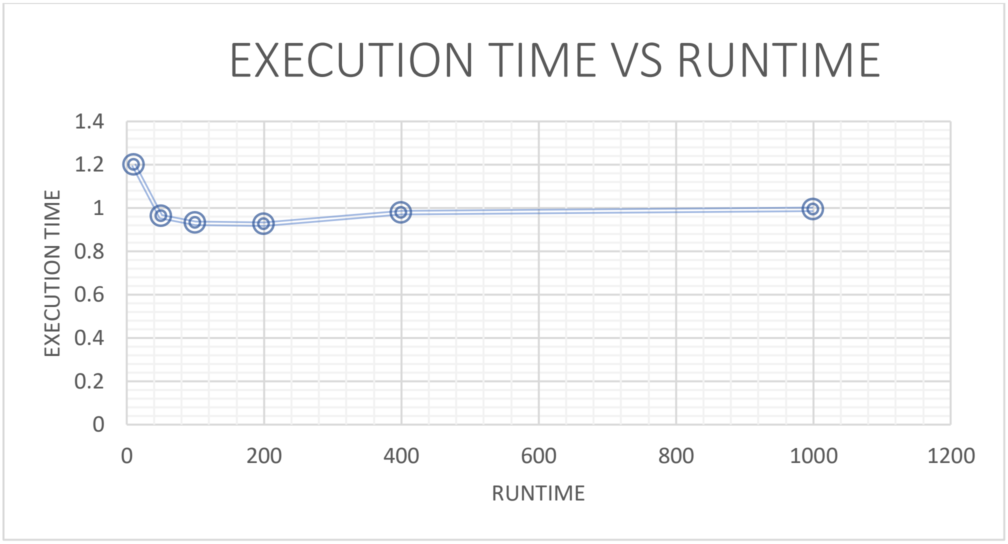
2) USE-CASE – To demonstrate a case for starvation, we create two processes, one with a priority of 20 and one with a priority of 100. Both the processes run an infinite loop. We create the processes with the lower priority first so that it is scheduled initially. However, we see that when the higher priority process is created it is always scheduled and the lower priority process is never scheduled again leading to starvation as the higher priority process is not getting killed.

PART – II (LOTTERY SCHEDULLING)

Files edited- clock.h, process.h, create.c,create_user_proc.c, resched.c, insert.c,kill.c

In lottery scheduling, all the user processes are assigned tickets by the user. This is done through the set tickets functions. This also increments a total Tickets counter. These tickets are indicative of the proportions of the CPU that each user process should receive. On the other hand, system processes should be scheduled with a higher priority. In this implementation, a single ready queue is used to store both, the system as well as the user processes. The priority of the user processes is set lower than system processes but greater than null process. The insert function is modified so that the user processes are inserted below system process in descending order of tickets. Thus, first all the system processes are scheduled with the default scheduling policy of XINU and xinu scheduling events are maintained. Once, we are out of the system processes to schedule, we schedule the user processes. The processes to be scheduled is decided by a random number lottery which depends on the number of tickets contained by each of the processes thus maintaining a proportional fair usage. This is implemented in resched.c. User processes are created in create_user_proc.c and their execution is controlled by the timed execution function implemented in timed_execution and clkhandler. Once a process is killed , its tickets are subtracted from the total tickets. This is implemented in kill.c

Fairness Analysis:



Run Time	Execution time P1	Execution Time P2	ETP1/ETP2
10	24	20	1.2
50	99	103	0.9611
100	190	204	0.9311
200	373	403	0.9255
400	785	803	0.9775
1000	1990	2002	0.99400

Thus, it is observed that when the runtime is small, the value of U = EPT1/EPT2 is greater than 1. But as the runtime increases, the value of U approaches 1. This indicates that as the runtime increases , the probability of fairness increases as ticket generation is random.

PART-III(MLFQ)

Files changed:
clock.h,process.h,initialize.c,create.c,create_user_proc.c,clkhandler.c,resched,c,ready.c

In this implementation, there are three user ready queues indicating three different priority level. When a process is created, it is inserted in the highest priority queue. This is implemented in ready.c. A q_num field is added to procent structure for this purpose. It indicates the queue that the user process belongs to. At the end of every time slice, a scheduling decision is made. If system processes are available , they are scheduled using xinu's default scheduling. If no system processes are available, the high priority user queue is checked first and then the subsequent user queues. This logic is implemented in resched.c . Once a process completes its allotted time, it downgrades. This logic is implemented in rearrange_queues in create_user_proc.c. Once the boost period is reached, all the processes are upgraded to the highest priority queues and their Time allotment is reset. Even the processes in the sleep queue are upgraded so that they wake up in the first queue. This is implemented in priority_boost in create_user_proc.c
Resched() calls are commented out in sleep and ready.