REPORT

Q1. What is the ready list? List all the system calls that operate on it.

Answer: Ready list is the list containing the list of processes that are in **ready state, i.e. eligible to get into running state**. It is arranged in the order of the priority with key as the priority number – uses doubly linked list.

System calls that operate on ready list:

1) Send
2) Signal
3) Wakeup
4) Resume
5) Signal

Q2. What is the default process scheduling policy used in Xinu? Can this policy lead to process starvation?Explain your answer.

Answer: Default scheduling policy used by Xinu is preemptive priority based scheduling and round robin scheduling with processes having equal priorities.

Yes, this process can lead to starvation of low priority jobs, as until the high priority jobs are completed, the lower priority task does not get a chance to execute.

Q3. When a context switch happens, how does the `resched` function decide if the currently executing process should be put back into the ready list?

Answer: The currently executing function is put back into ready **list if there exists a process with higher priority than the currently running process**, it puts the current function into ready state and then inserts it into ready list and dequeues the process with highest priority from the ready list.

Q4: Analyze Xinu code and list all circumstances that can trigger a scheduling event.

Answer: Instances:
1) When a new process is resumed after creation, scheduling takes place. in ready.c
2) When a process wakes up after sleep duration, scheduling take place. in wakeup.c
3) Once preemption is completed, every quantum, scheduling takes place- in clkhandler.c
4) When the currently running process has to be suspended it calls resched, - suspend.c
5) When a process has not yet received the message, it has to wait to terminate, it therefore triggers scheduling- receive.c
6) Once a running process is killed, resched is called- kill.c

Project report:

Q1: Preliminary coding tasks

Files of preliminary coding tasks present in Q1 folder in zipped file.

CTX1: Implement the function `syscall print_ready_list()` that prints the identifiers of the processes currently in the ready list.

File name- print_ready_list();

To print readylist, iterate over queue of ready list and print the data.

CTX2: Add PCB fields, runtime,turnaround time and num_ctxsw and update continuously.
Implementation:
In process.h files, the following fields are added:

- `uint32 runtime` – number of milliseconds the process has been running (i.e., in `PR_CURR` state).
- `uint32 turnaroundtime` – turnaround time in milliseconds.
- `uint32 num_ctxsw` – number of context switch operations *to* the process.

In clkhandler.c , runtime and turnaround time are incremented. Runtime is incremented if the user is in PR_CURR state.
Num_ctxsw is incremented in resched.c whenever context switch happens to a new process.

CTX3: create_user_process- this process is created and function definition is put in prototype file for external access.
The same code as create is used,except that priority is not taken as input and by default assigned as 0.

CTX4: Added DEBUG_CTXSW in resched function to give functionality to view context switch debugging for lottery scheduling purpose

CTX5: Implementation of burst_exectuion-
File name- system/burst_execution.c
Here, fields are added to PCB for burst execution to keep a tab on the bursts,
The clkhandler, decrements the burstduration everytime the user process is run and is in burst duration.
Once burstduration is completed, completed_burst flag is updated. This helps notifying burst_execution function that burst duration is over.

<u>Q2: Lottery Scheduling:</u>

User process to be scheduled are chosen based on a pseudo random generator , with highest priority given to user processes.
Files touched:

```
Mode            LastWriteTime       Length Name
----            -------------       ------ ----
-a----     10/12/2019   7:42 AM       1241 burst_execution.c
-a----     10/12/2019  10:15 AM       1677 clkhandler.c
-a----      9/30/2019   6:17 PM       1249 clkinit.c
-a----     10/12/2019  10:13 AM       3987 create.c
-a----     10/12/2019  10:14 AM       4064 create_user_process.c
-a----     10/12/2019  10:13 AM       6614 initialize.c
-a----     10/12/2019   1:10 PM       1308 kill.c
-a----     10/13/2019   4:27 AM       1907 main.c
-a----     10/12/2019   5:37 AM       1969 queue.c
-a----     10/12/2019  11:26 AM        977 ready.c
-a----     10/12/2019   5:20 PM       7712 resched.c


    Directory: C:\Users\ayush\Desktop\os\assignment2\q2\xinu\tempq2\include


Mode            LastWriteTime       Length Name
----            -------------       ------ ----
-a----      9/30/2019   6:17 PM        482 clock.h
-a----     10/11/2019  11:09 AM       2471 kernel.h
-a----     10/12/2019  10:16 AM       2743 process.h
-a----     10/12/2019   7:29 AM      15119 prototypes.h
-a----     10/10/2019  11:52 PM       1298 queue.h
```

Step1: number of queues- Added 2 to the total queue size in queue.h
Step2: added user queue in kernel.h
“extern   qid16   userproclist;”

Step 3: added tickets field and ticket flag in process.h
“uint32 tickets;   /* number of tickets for lottery scheduling */
uint32 ticketFlag;”

ticketFlag is used to indicate that the ticket has been updated and needs to be written to the key of the user queue.

Step4 : in inititalize.c the userproclist is initialized with new queue.

For lottery scheduling main code change is made in resched.c along with clkhandler.c

To implement lottery scheduling the following algorithm is used:

Reched.c

bool8 usr_proc;-- > indicates process is user process
uint32 tickets;   -> tickets assigned to the user process
uint32 ticketFlag; -> if there is any update to tickets, it gets set to 1

1) If the process that called the resched function is a user process, then check if any processes are present in ready list – as highest priority is system process. If there is a process present in ready list and the calling process is in running state, put the process to ready state, insert it to the userproclist, and dequeue the highest priority system process. If the calling process is not in running state, there is no need to put any process in the queue.Else, if there are no process except for null process in readylist check for user process in userproclist. If there is any process in user list, then call the lottery winner function to select the next process to run. Conditions to put process in ready state if it was currently running and put a process to current state if it is dequeued is taken care of. If there are no processes to run, then run null process
2) If the process calling the resched function is system process, check if there are any higher priority function that can be executed. If not, then give preference to user process, and similar procedure follows in the code, with careful check on each of the conditions:
    a. Is user process or system process?
    b. Is in running state or not?—if curr, put process to ready state and insert in readylist or userproclist depending on whether it is a system process or user process.
    c. Are there any high priority process is ready list or not, if not, is any user process available for lottery, else use null process.

Implementation of lottery_winner function:
total_tickets;- > updates the total number of tickets of all process in lottery_winner- global variable .

The lottery winner, iterates over userproclist, uses the tickets field of the process, and addes them up to the total number of tickets. Once total number of tickets is known rand function is used to compute the lottery winner. Winner is then passed back, and using **dequeue_userproclist** function defined in queue.c, process id eligible is found.

dequeue_userproclist in queue.c- iterates over queue keys , to find the pid of the matching key.

Set_tickets in create_user_process- sets the tickets and if the number of tickets is zero, updates the flag.Once the ticket value becomes equal to a value greater than zero, this flag gets updated in **clkhandler** to indicate, process is ready for use and calls ready function to resched it.

Kill.c→ turnaround time is updated.

Analysis of fairness of lottery scheduling :

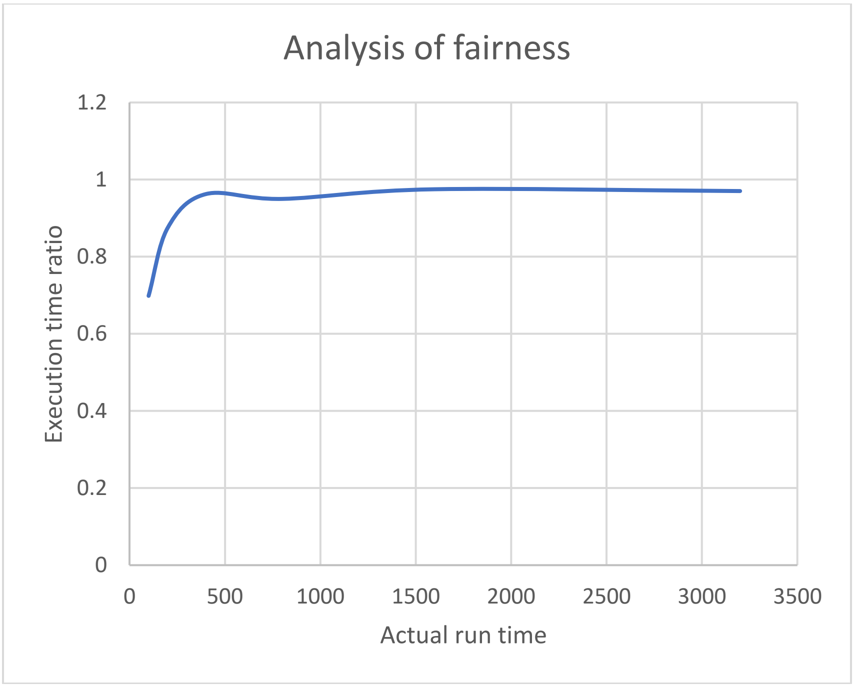
2 processes are run with same number of tickets=100, and for the same runtime at a go. This process is iterated for upto runtime starting form 100 going upto 4000

The code can be seen in the system/main.c file in the zip foder.

A graph is then plotted, to see the fairness of the lottery scheduling.
The date points are as below:

actual runtime	runtimeA	runtimeb	ratio
100	139	199	0.698492462
200	349	399	0.874686717
400	769	799	0.962453066
800	1519	1599	0.94996873
1600	3119	3199	0.974992185
3200	6209	6399	0.970307861

Graph:



Analysis of the fairness policy --
As indicated by the graph, as the runtime increases, the fairness of lottery scheduling increases. This is due to the probabilistic nature of lottery scheduling. Since the number of tickets assigned is the same, there is a 50 percent chance that process a will be picked up and 50 percent chance that process b will be picked up. The higher duration the process is run for ,more chances for the process to win a lottery fairly distributed.

Q3: Multi-level feedback queue scheduling:

In multi-level feedback queue, there are 3 levels of priority considered.

Here we use **FIFO queue** as data structure for implementing the 3 queues of different priority:
Within the queues, the list of processes are executed in round robin fashion – using the FIFO property of the queue. Amongst the 3 queues, 1st priority is always given to the queue with highest priority.

Files touched:

```
PS C:\Users\ayush\Desktop\os\assignment2\q3\xinu\tempq3\system> dir
    Directory: C:\Users\ayush\Desktop\os\assignment2\q3\xinu\tempq3\system

Mode             LastWriteTime         Length Name
----             -------------         ------ ----
-a----       10/12/2019  11:59 AM          1241 burst_execution.c
-a----       10/13/2019   3:38 AM          3160 clkhandler.c
-a----       10/13/2019   3:13 AM          1301 clkinit.c
-a----       10/12/2019   7:19 PM          4011 create.c
-a----       10/12/2019   2:57 PM          3783 create_user_process.c
-a----       10/12/2019  10:49 PM          6525 initialize.c
-a----       10/13/2019   3:28 AM          1270 kill.c
-a----       10/12/2019  11:53 PM          8497 main.c
-a----       10/13/2018   1:37 AM          1262 queue.c
-a----       10/12/2019  10:31 PM           922 ready.c
-a----       10/13/2019   5:14 AM          9693 resched.c

PS C:\Users\ayush\Desktop\os\assignment2\q3\xinu\tempq3\include> dir

    Directory: C:\Users\ayush\Desktop\os\assignment2\q3\xinu\tempq3\include

Mode             LastWriteTime         Length Name
----             -------------         ------ ----
-a----       10/13/2019   3:13 AM           481 clock.h
-a----       10/12/2019  10:31 PM          2502 kernel.h
-a----       10/12/2019   2:56 PM          2516 process.h
-a----       10/12/2019  12:00 PM         15077 prototypes.h
-a----       10/12/2019   7:51 PM          1284 queue.h
-a----       10/13/2019   5:16 AM           508 resched.h
```

Step1: In queue.h -- > number of files changed, resched.h- defined- TIME_ALLOTMENT, PRIORITY_BOOST_PERIOD, in process.h defined-

uint32 qnum; → which priority queue the process belongs to, initialized to 1- highest priority in initialize.c

uint32 time_used;→ time used tracking for checking with allotted time

extern   uint32   boost;   /*boost time elapsed to check with the priority boost time elapsed*/

Step2: kernel.h- > queues are defined -all 3 lists:
extern   qid16 user_priority_list1;
extern   qid16 user_priority_list2;
extern   qid16 user_priority_list3;
This is initialized in initialize.c

Step 3: clkhandler.c → assignment of preempt is removed from clkhandler as it will be taken care of in resched,c, boost time is incremented and compared with PRIORITY_BOOST_PERIOD,when that is reached, priority_boost function is called ,in order to reset the time allotment and move the queues to 1st queue. This is done inclusive of the current process and sleep queue.
Once the preempt assigned in resched becomes 0, resched is called again. time_used is incremented for a process and the check for TIME_ALLOTMENT is made.

Step 4: ready.c- > Whenever the user creates a user function and resumes the process, it hits the ready file. Here a check for user process is made, and if it is userprocess, qnum of the process is checked and is enqueued in respective queue.

Step 5: resched.c -> here the main implementation of MLFQ is done.
An array is maintained to store the list name and is iterated in the code to find the processes in the queue and dequeue it.
"qid16 array_userlist[3]={user_priority_list1,user_priority_list2,user_priority_list3};"
Following checks are made:

1) If the current running process is a system process, check the readylist for other user process, if present, schedule it by dequeueing it and enqueing running process to ready list after changing state to READY. If there is no system process, check for user process, all three lists in order of priority. If any of the queue has a process, then that process is run else if no processes are present, null process is run.
2) If the current running process is a user process, a check for for TIME_ALLOTMENT is made. If time_used has exceeded TIME_ALLOTMENT, then the process is moved to a lower priority queue and the time_used is reset. The process is moved to ready state and further checks for availability of system process, followed by availability of user process is made.

If the user process is present in the highest priority queue, it is dequeued from the queue and put to CURR state.

----------------------------END OF REPORT---