Swapping

Pseudocode

1. Initialize the queues (normal job dispatch queue, real time job dispatch queue, level 0 queue, level 1 queue, level 2 queue), initialise variables accumulated_arrival_times, accumulated_service_times, accumulated_finish_times, n_process, set initial value = 0.
2. Initialize the simulated memory block with size = 1024 (allocate memory on heap), offset = 0 and allocated = 0
3. Fill job dispatch queue from job dispatch file, with real-time jobs being loaded in the RT job dispatch queue, and normal jobs loaded in the Normal job dispatch queue
4. Ask the user to enter an integer value for time_quantum
5. While there is currently running processes or either queue is not empty
    I. Set check_job_addmission = 0 *(usage: require consider admission of arrived jobs in the arrived job queues immediately or not)*
    II. Unload any arrived pending process from the job dispatch queue
        A. dequeue process from RT Job Dispatch Queue and enqueue on level 0 queue
        B. dequeue process from normal Job Dispatch Queue and enqueue on level 1 queue
    III. If there is a currently running process and the process's allocated time has expired:
        A. Decrement the process's remaining_cpu_time by quantum;
        B. Terminate the process: send SIGINT to the process
        C. Increasing accumuate_service_times and accumutaed_arrival_times based on current process information. Also Increasing accumulated_finish_times by current timer. Increase n_process by 1.
        D. Deallocate the memory block's memory
        E. Deallocate the PCB (process control block)'s memory
        F. Set the current running process as null
        G. Set check_job_addmission = 1, which means that need to consider admission of arrived jobs in the arrived job queues immediately
    IV. job admission check and enqueue available jobs to level 1 queue
    Check real time job admission
        A. IF arrived RT job queue is empty, check normal job admission:
            1) While the normal job dispatch queue is not empty
            2) check if memory can be allocated using first fit allocation scheme
            3) If can find appropriate location, dispatch job
            4) Else break the while loop since we cannot find appropriate location for the next job

B. Else if arrived RT job queue is not empty, check RT job admission:
    1) check if there is enough block with size no less than the size of first RT job using first fit allocation scheme
    2) If there is not enough memory to fit the arrived real time job and there is some level 2 jobs either in level 2 queue or currently running
        a. If level 2 queue is not empty
            a) find the longest remaining CPU time job in L2 queue
            b) if current running process also a level 2 job
            c) compare its remaining CPU time with other jobs, and set the longest_L2_job as the current_process if the remaining cpu time of current running process is longer than that found in level 2 queue
        b. else, level 2 queue is empty, and thus the current running level 2 job is the longest remaining CPU time job, which will be swapped out

V. If longest_L2_job is not null, which means that there is blocking real-time job, we need to swapped out the level 2 job with the longest remaining CPU time
    A. if the longest_L2_job is the currently running process
        a. Decrement the process's remaining_cpu_time by quantum;
        b. Suspend the currently running process
        c. set the current running process as null
    B. else, the longest_L2_job is from the level 2 queue, remove the longest level 2 job from the level 2 queue
    C. push the longest_L2_job to swap queue
    D. Deallocate the PCB (process control block)'s memory
    E. Set the longest_L2_job as null

VI. swap in the arrived RT time jobs
    A. While the RT_job_dispatch_queue is not empty
    B. check if memory can be allocated using first fit allocation scheme
    C. if the required memory can be applicated, dispatch job
    D. Else break the while loop since we cannot fit more real time jobs

VII. If there is a currently running process:

    A. Decrement the process's reining_cpu_time by quantum;
    B. If the current running process cannot finish within the time_quantum and either L1 queue or L2 queue is not empty
        1) Suspend the currently running process: send SIGSTP to the process
        2) Add this process to the L2 queue
            a. append job to the end of L2 queue, if priority is 1, and modify priority = 2
            b. append job to the front of L2 queue, if priority is 2
    C. Else if the current running process with priority = 1

1) Set check_job_addmission = 1, which means that need to consider admission of arrived jobs in the arrived job queues immediately
2) Suspend the currently running process: send SIGSTP to the process
3) append job to the end of L2 queue, and modify priority = 2

D. Set the current running process as null

VIII. If we don't need to check job admission immediately, and there is no running process and there is a process ready to run(level 0 or level 1 or level 2 queue or swap job queue is not empty):

A. If level 0 queue is not empty, dequeue the process at the head of the level 0 queue and set it to current_process
B. Else if level 1 queue is not empty, dequeue the process at the head of the level 1 queue and set it to current_process
C. Else if level 2 queue is not empty, dequeue the process at the head of the level 2 queue and set it to current_process
D. Else if both normal job dispatch queue and real time job dispatch queue are empty,
    1) dequeue the process at the head of the swapped job queue and set it to current_process
    2) allocate new memory block for this process
E. If the process job is a suspended process, send SIGCONT signal to resume it
F. Else start it and set its status as running

IX. Calculate the time_quantum

A. If there is current running process:
    1) For the real-time job with priority = 0, quantum is its remaining_cpu_time
    2) for the normal job with priority = 1, quantum is the minimum value of time_quamtum and remaining_cpu_time
    3) for the level 2 jobs or the swapped in job, set the quantum = 1, and keep check every second
B. If the job dispatch queue is not empty, but both normal job dispatch queue and RT job dispatch queue is empty, and multi-level queues are all empty
    1) if either normal job dispatch queue or RT job dispatch queue is not empty and multi-level queues are all empty, set quantum = 1
    2) Else set quantum = 0

X. Let the dispatcher sleep for quantum;

XI. Increment the dispatcher's timer;

XII. Go back to Step 4.

6. Calculate and print the average turnaround time and the average waiting time
7. Free the memory of memory block and then terminate the job dispatcher

To implement the above algorithm, we need

1. two more PCB functions to find the process with the maximum remaining CPU time and able the remove specific job from the queue based on the processID.
2. Three more MAB functions to check memory available, allocate memory, and free memory

PcbPtr findMaxRemainingTime(PcbPtr headofQ)

- find the job with the longest remaining execution time based on the first fit rule
- returns:
  - PcbPtr: node with the longest remaining cpu time
  - NULL if queue was empty

PcbPtr removeSpecificJob(PcbPtr hPtr, PcbPtr target)

- remove the target node from the queue based on pid
- returns:
  - PcbPtr: the header of queue after remove the target node
  - NULL if queue was empty

MabPtr memChk(MabPtr arena, **int** size)

- check for memory available
- returns address of "First Fit" block or NULL

MabPtr memAlloc(MabPtr arena, **int** size)

- allocate a memory block
- returns address of block or NULL if failure

MabPtr memFree(MabPtr m)

- de-allocate a memory block
- returns address of block or merged block

Test case

General cases need to be tested:

- Test current-running level 2 job can be swapped out (test case 1, 2, 3)
- The swapped level 2job must be the one which has the **longest remaining execution time** (including both the blocked level 2 jobs in the level 2 queue and the current running level 2 process) (test case 5)
- When a level 2 job is swapped out, its associated memory must be freed. (test case 1, 2, 3, 5)
- The jobs in the swapped job queue can only be swapped in and scheduled to run again only when all the arrived job queues and the three-level ready queue are empty. (test case 1, 2, 3, 5)
- If the arrived normal job cannot fit in memory, it will be blocked until some jobs (level 0, level 1 or level 2) are terminated. (test case 1, 5)

- **Test the FCFS behavior of real-time job**: If the earlier arrival RT job is blocked due to the memory constraints, all following RT jobs must be blocked, even if there exists a free memory block for the RT job with smaller memory requirements. (all test cases)
- **Test level 1 queue:** No jobs in the level 1 queue can run more than one time_quantum, and the level 1 queue run in a FCFS manner (test case 1)
- **Test the FSFC behavior of normal job in level 1 and level 2 queue**: (test case 1)
- **Test the FCFS behavior of the swapped job queue and** when being able to swap in, they can only be swapped in **one** at a time. (test case 5)
- The level 2 job can be scheduled to run when both level 0 and level 1 are empty (test case 5)
- **Test level1 queue:** the normal job with priority = 1, can run time_quantum **without interpretation** within the given time_quantum even if a real-time job arrives during the quantum period (test case 2)
- **Test level 2 queue:** Test when a new job arrives (RT job or normal job) and can fit in memory, the currently running level-2 job will be preempted and placed in the front of the level-2 queue. And when the level-2 job can be scheduled again, this job will re-start before other level-2 jobs. (test case 5)
-

Special cases considered:

- **Test empty system behavior**: when two dispatch queues and multiple level queues are empty, but there are still some unarrived jobs, the system will wait for the unarrived jobs, and schedule them to run until they arrive. (test case 1, 2, 3, 4)
- The level 2 queue is empty, but the currently running job has priority = 2, then when a real-time job comes, it should be suspended immediately, swapped out if not enough memory. (test case 1, 5)
- In some situation, level 2 job can be swapped out immediately, even if it just swapped in level 2 queue (for example, level 1 job can be pushed to level 2 queue at the end of timer = 11, then swapped out before timer = 12 due to the memory constraints) (test case 2, 3)
- **Test the swapped out job can be scheduled to run during the middle of the program with the FCFS order:** the swapped out job can be swapped in to run again when there are "unarrived job remaining", but two dispatch job queues and the 3 level job queues are empty. (test case 1, 2, 3)
- In some case when time_quantum is very large, all jobs can finish within only one time_quantum. We design this test case to test the normal job with priority = 1 will not be interrupted within the time_quantum, even if a real-time job arrives during the quantum period. (test case 6)

(My program gives the the expected result based all test cases below)

Job Dispatch List 1

0, 12, 1, 1000 (job 1)
3, 7, 1, 180 (job 2)
10, 6, 0, 320 (job 3)
12, 5, 0, 245 (job 4)
14, 3, 1, 65 (job 5)
17, 6, 1, 32 (job 6)
43, 3, 0, 32 (job 7)

We design this situation:

- Only one job (the job arrived at timer = 0) can be swapped out, and it is the current running process that needs to be swapped out.
- The real-time job can be scheduled to run immediately, given the current process is level2 jobs. If the memory of real-time jobs cannot fit, the level 2 job with the largest remaining CPU time will be swapped out.
- Given time_quantum = 10 or 11, level 2 job can be swapped out immediately, even if it just swapped in level 2 queue (for example, level 1 job can be pushed to level 2 queue at the end of timer = 11, then swapped out before timer = 12 due to the memory constraints)
- **Test empty system behavior**: when two dispatch queues and multiple level queues are empty, but there are still some unarrived jobs, the system will wait for the unarrived jobs, and schedule them to run until they arrive. (The system will be empty few seconds before the last real-time job (arrived at timer = 43) is scheduled to run)
- **Test the swapped out job can be scheduled to run during the middle of program with the FCFS order:** the swapped out job can be swapped in to run again when there are "unarrived job remaining", but two dispatch job queues and the 3 level job queues are empty.

Test case 1: Job dispatch list 1 + time_quantum = 2

- From timer = 0 to timer = 10, only the first arrived normal job runs (since timer = 3, the priority = 2), as the normal job arrived before timer 10 cannot fit the memory. And this job will be swapped in at the timer = 38 (with offset = 0) after all other jobs terminate.
- The normal job arrived at timer = 3, will be scheduled to run time_quantum after all real-time job terminate.
- At timer = 10, the real-time job will be scheduled to run with offset = 0 as the level 2 job swapped out.
- The normal job arrived at timer = 14 and 17 will be scheduled to run after the normal job arrived at timer = 3 run one time_quantum, based on FCFS rule.

Expected result for test case 1

The average turnaround time is: 18.000000
The average waiting time is: 12.000000

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	offset	Action at the end of quantum
0-2	1	1	0	12	2	0	Pushed to level 2 queue
-10	1	2	0	10	8	0	Swapped out at timer = 10
-16	3	0	10	6	6	0	Finish and terminate
-21	4	0	12	5	5	0	Finish and terminate
-23	2	1	3	7	2	245	Pushed to end of level 2 queue
-25	5	1	14	3	2	425	Pushed to end of level 2 queue
-27	6	1	17	6	2	0	Pushed to end of level 2 queue
-32	2	2	3	5	5	245	Finish and terminate
-33	5	2	14	1	1	425	Finish and terminate
-37	6	2	17	4	4	0	Finish and terminate
-39	1	Swapped in	0	2	2	0	Swapped in at timer = 38, then run 2 sec. Then finish and terminate
-43	-	-	-	-	-	-	Empty system
-46	7	0	43	3	3	0	Finish and terminate

Test case 2: Job dispatch list 1 + time_quantum = 10

- From timer = 0 to timer = 10, only the first arrived normal job runs (since timer = 3, the priority = 2), as the normal job arrived before timer 10 cannot fit the memory. And this job will be swapped in at the timer = 38 (with offset = 0) after all other jobs terminate.
- The normal job arrived at timer = 3, will be scheduled to run time_quantum after all real-time job terminate.
- At timer = 10 , the first normal job finish run the first quantum and pushed to the level 2 queue.

Expected result for test case 2

The average turnaround time is: 17.142857

The average waiting time is: 11.142858

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	Offset	Action at the end of quantum
0-10	1	1	0	12	10	0	Swapped out at timer = 10
-16	3	0	10	6	6	0	Finish and terminate
-21	4	0	12	5	5	0	Finish and terminate
-28	2	1	3	7	7	245	Finish and terminate
-31	5	1	14	3	3	425	Finish and terminate
-37	6	1	17	6	6	0	Finish and terminate
-39	1	Swapped in	0	2	2	0	Swapped in at timer = 38, then run 2 sec. Then finish and terminate
-43	-	-	-	-	-	-	Empty system
-46	7	0	43	3	3	0	Finish and terminate

Test case 3: Job dispatch list 1 + time_quantum = 11

- **Test level1 queue:** the normal job with priority = 1, can run time_quantum **without interpretation** within the given time_quantum even if a real-time job arrives during the quantum period.
- The level1 job will be swapped out at timer = 11. That is, this job will be treated as a level 2 job immediately at the end of timer =11. But since real-time arrives at timer = 10 and not enough memory to fit RT job, this level 2 job will be swapped out at the end of timer = 11.
- The remaining logic should be the same as tests case 2.

Expected result for test case 3

The average turnaround time is: 17.857143

The average waiting time is: 11.857142

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	offset	Action at the end of quantum
0-11	1	1	0	12	11	0	Swapped out at timer = 11
-17	3	0	10	6	6	0	Finish and terminate

-22	4	0	12	5	5	0	Finish and terminate
-29	2	1	3	7	7	245	Finish and terminate
-32	5	1	14	3	3	425	Finish and terminate
-38	6	1	17	6	6	490	Finish and terminate
-39	1	Swapped in	0	2	2	0	Swapped in at timer = 39, then run 2 sec. Then finish and terminate
-43	-	-	-	-	-	-	Empty system
-46	7	0	43	3	3	0	Finish and terminate

Test case 4: Job dispatch list 1 + time_quantum = 12

In this case, both the level 2 job queue and the switch job queue are empty all the time, as all jobs can finish within one time quantum.

Expected result for test case 4

The average turnaround time is: 14.571428

The average waiting time is: 8.571428

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	Offset	Action at the end of quantum
0-12	1	1	0	12	12	0	Finish and terminate
-18	3	0	10	6	6	0	Finish and terminate
-23	4	0	12	5	5	320	Finish and terminate
-30	2	1	3	7	7	565	Finish and terminate
-33	5	1	14	3	3	0	Finish and terminate
-39	6	1	17	6	6	65	Finish and terminate
-43	-	-	-	-	-	-	Empty system
43-46	7	0	43	3	3	0	Finish and terminate

Job Dispatch List 2

0, 7, 1, 70 (job 1)
2, 2, 0, 32 (job 2)
4, 10, 1, 900 (job 3)
6, 6, 1, 32 (job 4)
9, 7, 0, 32 (job 5)
18, 9, 1, 880 (job 6)
19, 3, 1, 60 (job 7)
26, 2, 0, 32 (job 8)

We design this situation:

- There are multiple jobs in the swapped job queue
- The swapped job queue will be scheduled to run one by one in FCFS behavior
- The swapped out level 2 job is the one with the longest remaining CPU time (compare both the current running level 2 job and the jobs block within level 2 queue
- If the arrived normal job cannot fit in memory, it will be blocked until some jobs (level 0, level 1 or level 2) are terminated.
- The level 2 job can be scheduled to run when both level 0 and level 1 are empty. And if need to push back again to the level 2 queue, it will be at the front of the queue.

Test case 5: Job dispatch list 2 + time_quantum = 2

- Using time quantum = 2, all normal jobs cannot finish within time quantum, thus there will be multiple jobs in the level 2 queue
- The level 2 job can be scheduled to run when both level 0 and level 1 are empty, thus, during timer = 9, 16-18, the first arrived noral job is scheduled to run, and since cannot finish, wil be pushed back to the front of the level 2 queue.
- At timer between 22 and 24, the normal job arrived at timer = 19 with priority = 1 is scheduled to run for 2 seconds, since a level 2 job terminates at the end of timer = 22, so freed memory can fit this black normal job.
- At timer = 9, since new real-time jobs arrive, but not enough memory left, the level 2 job with the longest remaining CPU time (job 3) is swapped out.
- At timer = 26, since new real-time jobs arrive, but not enough memory left, the level 2 job with the longest remaining CPU time (job 6) is swapped out.
- The job 3 and job 6 will be run when the whole system is empty. In this case, job 3 and 6 will be scheduled to run one by one with offset = 0 at the end. And job 3 will be scheduled first based on the FCFS rule.

Expected result for test case 5

The average turnaround time is: 16.500000

The average waiting time is: 10.750000

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	Offset	Action at the end of quantum
0-2	1	1	0	7	2	0	Pushed to end of level 2 queue
-4	2	0	2	2	2	70	Finish and terminate
-6	3	1	4	10	2	70	Pushed to end of level 2 queue
-8	4	1		6	2	970	Pushed to end of level 2 queue
-9	1	2		5	1	0	Push to the front of level 2 queue
-	-	-	-	-	-	-	The job 3 swapped out to the swapped queue at timer = 9
-16	5	0	9	7	7	70	Finish and terminate
-18	1	2	0	4	2	0	Push to the front of level 2 queue
-20	6	1	18	9	2	70	Pushed to end of level 2 queue
-22	1	2	0	2	2	0	Finish and terminate
-24	7	1	19	2	2	0	Pushed to end of level 2 queue
-26	4	2	6	4	2	970	Push to the front of level 2 queue
-	-	-	-	-	-	-	The job 6 swapped out to the swapped queue at timer = 26
-28	8	0	26	2	2	60	Finish and terminate
-30	4	2	28	2	2	970	Finish and terminate
-31	7	2	19	1	1	0	Finish and terminate
-39	3	Swapped in job	4	8	8	0	Swapped in at timer = 31, Finish and terminate at timer = 39
-46	6	Swapped in job	18	7	7	0	Swapped in at timer = 39, Finish and terminate at timer = 46

Test case 6: Job dispatch list 2 + time_quantum = 10

Using time_quantum = 10, all jobs can finish within only one time_quantum. We design this test case to test the normal job with priority = 1 will not be interrupted within the time_quantum, even if a real-time job arrives during the quantum period.

Expected result for test case 6

The average turnaround time is: 15.625000
The average waiting time is: 9.875000

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	offset	Action at the end of quantum
0-7	1	1	0	7	7	0	Finish and terminate
-9	2	0	2	2	2	0	Finish and terminate
-16	5	0	9	7	7	0	Finish and terminate
-26	3	1	4	10	10	32	Finish and terminate
-28	8	0	26	2	2	0	Finish and terminate
-34	4	1	28	6	6	932	Finish and terminate
-43	6	1	18	9	9	32	Finish and terminate
-46	7	1	19	2	2	964	Finish and terminate

Job Dispatch List 3

0, 12, 1, 32
3, 7, 1, 32
10, 6, 0, 32
12, 5, 0, 32
14, 3, 1, 32
17, 6, 1, 32
43, 3, 0, 32

We design this situation:

- The sum of memory requirement of all processes is less than the simulator memory (1GB). In this simulation, the results using whatever time_quantum should have the exact same result at stage 1 and stage 2

- **Test empty system behavior**: when multiple level queues are empty, but there are still some unarrived jobs, the system will wait for the unarrived jobs, and schedule them to run until they arrive. (The system will be empty few seconds before the last real-time job (arrived at timer = 43) is scheduled to run)

Test case 7: Job dispatch list 1 + time_quantum = 2

- Expect the same scheduling result as the test case 1 of stage 1

Test case 8: Job dispatch list 1 + time_quantum = 6

- Expect the same scheduling result as the test case 2 of stage 1

Test case 9: Job dispatch list 1 + time_quantum = 12

- Expect the same scheduling result as the test case 3 of stage 1