Simple Memory Management

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 0 or 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) While the RT 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

V. If there is a currently running process:
   A. Decrement the process's remaining_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) Suspend the currently running process: send SIGSTP to the process
      2) append job to the end of L2 queue, and modify priority = 2
   D. Set the current running process as null

VI. If we donot need to check job addmission immediately, and there is no running process and there is a process ready to run(level 0 or level 1 or level 2 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. If the process job is a suspended process, send SIGCONT signal to resume it
   E. Else start it and set its status as running

VII. 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 normal job with priority = 2, 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

VIII. Let the dispatcher sleep for quantum;

IX. Increment the dispatcher's timer;

X. 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 three more MAB functions to check memory available, allocate memory, and free memory:

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:

1. **Test level 0 queue**: **real-time jobs** can run in a FCFS manner **without interruption** before completion. (all test cases)
2. **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)
3. **Test the FSFC behavior of normal job in level 1 and level 2 queue** (all test cases)
1. **Test memory merges and job submission:** When a job is terminated, the relative memory should be freed immediately, and the system can consider the admission of arrived jobs **immediately**. (all test cases)
2. **Test level 2 queue:** Test when a new job arrives (RT job or normal jobs) and there is enough available memory, the currently running level-2 job will be preempted be placed in the front of the level-2 queue. (test case 3)

3. **Test level 2 queue:** Test when a new job arrives (RT job or normal jobs) but there is not enough available memory, the currently running level-2 job will continue running, and the new arrived job will be blocked until enough available memory is released. (test case 1)
4. **Test normal job with long CPU time**: Test when a new job arrives (RT job or normal jobs) and there is enough available memory, if the currently running jobs with priority = 1 cannot finish within one time_quantum, then it will be pushed on the level 2 queue. (test case 1)
5. **Test the offset of memory location:** When some memory released, and the following process needs the memory size less than the released memory size, based on the **first fit** rule, the new process will be please as the same offset as the last terminated process (all test cases)
6. **Test memory split**: Only allocate required size of memory for each job, can be checked based on the printed offset (all test cases)
7. **Test memory free**: The memory will be freed immediately after the job terminated. (all test cases)
8. **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 4)

Special cases considered:

1. The sum of memory requirement of all processes is less than the simulator memory (1GB) (test case 5, 6, 7)
2. The sum of memory requirement of all processes is more than the simulator memory (1GB) (test case 1)
3. When a job is terminated, and there is real-time job admission, the job blocked in the level2 queue shouldn't run in between. (test case 1, 2)
4. Due to the memory constraints, the real-time jobs might be blocked until some normal (level 2)job finish. (test case 1)
5. Due to the memory constraints, the real-time jobs might be blocked until some normal (level 1) job free memory if time_quantum big enough for normal job finish within one time_quantum (test case 2)
6. Test memory can merge with the previous continuous free block (test case 1, 3)
7. Test memory can merge with the following continuous free block (all test cases)
8. when normal job arrive at the same time with real-time job, then real-time job start first (test case 3, 4)
9. 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 4)
10. **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 (test case 5, 6, 7)

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

Job Dispatch List 1

0, 12, 1, 32 (job 1)
2, 2, 1, 180 (job 2)
4, 6, 0, 320 (job 3)
6, 5, 0, 245 (job 4)
7, 3, 1, 300 (job 5)
8, 4, 1, 300 (job 6)
17, 4, 0, 420 (job 7)
18, 4, 0, 600 (job 8)
19, 4, 0, 420 (job 9)

We design this situation:

- The sum of all memory requirements is more than the simulator memory (1GB), so the last arrival real-time job (arrive at timer = 17) will be blocked until enough memory is released.
- All real-time jobs should run without interruption before terminating.

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

- When real-time job admission (arrive at timer = 4) at timer = 4, the level 2 jobs (the normal job arrived at timer = 0 in this case) should not re-start since the real-time job has higher priority.
- The sum of all memory requirements is more than the simulator memory (1GB), so the real-time job (arrive at timer = 17) will be blocked until the jobs arrive before timer 8 finishes.
- Given time_quantum = 2, 3 normal job (arrive at timer =0, 7, 8) will be pushed on the level 2 queue.
- The offset of the real-time job (arrive at timer = 17) should be 0, as the previous freed block will be merged automatically.
- The real-time job arrived at timer = 18 will be blocked until all previous level0, level1, and level 2 job finish, due to the memory constraints (the normal job arrives at timer 8 should be placed at offset = 577, so there is not enough memory block to fit the next real-time, which needs 600 bytes, before this normal job terminates)
- The RT job arrived at timer = 19 will be blocked until the RT job arrives at timer = 18 finish based on the FCFS rule.

Expected result for test case 1

The average turnaround time is: 17.888889
The average waiting time is: 13.000000

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	Push to the end of level 2 queue
-4	2	1	2	2	2	32	Finish and terminate
-10	3	0	4	6	6	32	Finish and terminate
-15	4	0	6	5	5	32	Finish and terminate
-17	5	1	7	3	2	277	Push to the end of level 2 queue
-19	6	1	8	4	2	577	Push to the end of level 2 queue
-29	1	2	0	10	10	0	Finish and terminate
-30	5	2	7	1	1	277	Finish and terminate
-34	7	0	17	4	4	0	Finish and terminate
-36	6	2	8	2	2	577	Finish and terminate
-40	8	0	18	4	4	0	Finish and terminate
-44	9	0	19	4	4	600	Finish and terminate

Test case 2: Job dispatch list 1 + time_quantum = 12

Using time_quantum = 12, all jobs can finish within only one time_quantum. So this case can help to test job priority more easily

- Based on the arrival time setting, the first normal job should run before the real-time job starts
- The normal job arrived at timer = 2, will be allocated memory after the first normal job terminate at timer = 12 (after allocating all real-time jobs arrived before timer = 12, there is still enough memory block for this normal job, so it can be allocated after the first normal job terminate), but it will be blocked until the real-time job arrived at timer = 17 terminates.
- The last two real-time jobs will start after the normal job arrives at timer = 2 finishes. Since there is not enough memory block for the job arrived at timer = 18, so this normal job has to finish and free memory first.
- The last two arrived normal jobs will be blocked until all real-time jobs terminate based on the priority order.

Expected result for test case 2

The average turnaround time is: 20.222221

The average waiting time is: 15.333333

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	4	6	6	0	Finish and terminate
-23	4	0	6	5	5	320	Finish and terminate
-27	7	0	17	4	4	0	Finish and terminate
-29	2	1	2	2	2	565	Finish and terminate
-33	8	0	18	4	4	0	Finish and terminate
-37	9	0	19	4	4	600	Finish and terminate
-40	5	1	7	3	3	0	Finish and terminate
-44	6	1	8	4	4	300	Finish and terminate

Job Dispatch List 2

0, 8, 1, 32 (job 1)
0, 8, 0, 32 (job 2)
2, 7, 1, 500 (job 3)
4, 4, 1, 400 (job 4)
5, 9, 0, 100 (job 5)
6, 4, 0, 32 (job 6)
25, 4, 0, 32 (job 7)

We design this situation:

- Test when normal job arrive at the same time with real-time job, then real-time job start first
- Test when all real-time require a relatively small memory block, and can always be fit in memory after the previous real-time job terminates, then they will start before the normal job.

Test case 3: Job dispatch list 2 + time_quantum = 1

- Except the last arrived real-time jobs, all other real time jobs will run and terminate before the first normal job start, since all real-time require relative small memory block, and can always be fit in memory after previous real-time job terminate
- Using time_quantum = 1, the normal job arrived at timer = 4 can only run after the previous normal jobs (arrived at timer = 0 and timer = 2) terminate, free and merge memory blocks.

- The offset of the last normal job is 0, as when the last normal job starts, all previous jobs should be terminated, free and merge the whole memory block.
- The normal job arrived at timer = 0 will be prememped at the front of level 2 queue, when the real-time job arrived at timer = 25 arrive. And restart after the real-time job arrived at timer = 25 finish.

Expected result for test case 3

The average turnaround time is: 21.571428

The average waiting time is: 15.285714

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	offset	Action at the end of quantum
0-8	2	0	0	8	8	0	Finish and terminate
-17	5	0	5	9	9	0	Finish and terminate
-21	6	0	6	4	4	100	Finish and terminate
-22	1	1	0	8	1	132	Push to the end of level 2 queue
-23	3	1	2	7	1	164	Push to the end of level 2 queue
-25	1	2	0	7	2	132	Push to the front of level 2 queue
-29	7	0	25	4	4	0	Finish and terminate
-34	1	2	0	5	5	132	Finish and terminate
-40	3	2	2	6	6	164	Finish and terminate
-41	4	1	4	4	1	0	Push to the end of level 2 queue
-44	4	2	4	3	3	0	Finish and terminate

Test case 4: 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. Thus, the real-time job arrived at timer = 25 will start after the normal job arrives at timer = 0 finish.

Expected result for test case 4

The average turnaround time is: 21.428572

The average waiting time is: 15.142858

Timer	Job	Priority	Job arrive time	Remaining Cpu time	Quantum	offset	Action at the end of quantum
0-8	2	0	0	8	8	0	Finish and terminate
-17	5	0	5	9	9	0	Finish and terminate
-21	6	0	6	4	4	100	Finish and terminate
-29	1	1	0	8	8	132	Finish and terminate
-33	7	0	25	4	4	0	Finish and terminate
-40	3	1	2	7	7	164	Finish and terminate
-44	5	1	4	4	4	0	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
- **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 5: Job dispatch list 1 + time_quantum = 2

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

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

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

Test case 7: Job dispatch list 1 + time_quantum = 12

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