Sameer Gupta
OS Lab
November 21, 2025

Practical 8

Question 1. Write a CPP program to simulate the CPU scheduling algorithm MLQ(Multilevel Queue).

```cpp
#include<bits/stdc++.h>
using namespace std;

void bubbleSort(vector<int>&at, vector<int>&bt, vector<int>&pid, vector<int>&prio){
    int n=at.size();
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-1;j++){
            if(at[j]>at[j+1] || (at[j]==at[j+1] && pid[j]>pid[j+1])){
                swap(at[j],at[j+1]); swap(pid[j],pid[j+1]);
                swap(bt[j],bt[j+1]); swap(prio[j],prio[j+1]);
            }
        }
    }
}

int getPID(vector<int>&bt, vector<int>&at, vector<int>&remProcess, int currTime, vector<int>&remTime
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;
    for(int pid : remProcess){
        int i = pid - 1;
        // Process must be Priority 3 (Batch), Arrived, and Not Completed
        if(at[i]<=currTime && remTime[i]>0 && prio[i] == 3){
             pq.push({remTime[i], at[i]*1000 + pid});
        }
    }
    if(pq.empty()) return -1;
    return pq.top().second % 1000;
}

int main(){
    cout<<"--- MLQ SCHEDULER ---"<<endl;
    int n; cout<<"Enter number of processes: "; cin>>n;
    vector<int>pid(n), at(n), bt(n), prio(n), ct(n), tat(n), wt(n);
    vector<float>pr(n);
    for(int i=0;i<n;i++){ cout<<"AT P"<<i+1<<": "; cin>>at[i]; pid[i]=i+1; }
    for(int i=0;i<n;i++){ cout<<"BT P"<<i+1<<": "; cin>>bt[i]; }
    for(int i=0;i<n;i++){ cout<<"Prio P"<<i+1<<" (1=Sys, 2=Int, 3=Bat): "; cin>>prio[i]; }

    bubbleSort(at, bt, pid, prio);

    vector<int>remTime(bt);
    vector<int>remProcess; for(int x:pid) remProcess.push_back(x);
    vector<int>slice(n,0);
    vector<bool>in_q(n, false);

    queue<int> q0, q1; // Store Indices directly
    int completed=0, currTime=at[0];

    // Initial load
    for(int i=0; i<n; i++){
        if(at[i]<=currTime){
```

```
            if(prio[i]==1) { q0.push(i); in_q[i]=true; }
            else if(prio[i]==2) { q1.push(i); in_q[i]=true; }
        }
    }

    while(completed<n){
        int idx = -1, type = 0; // 1=Q0, 2=Q1, 3=Q2

        // Priority Check
        if(!q0.empty()){ idx=q0.front(); type=1; }
        else if(!q1.empty()){ idx=q1.front(); type=2; }
        else {
            int id = getPID(bt, at, remProcess, currTime, remTime, prio);
            if(id!=-1) {
                for(int i=0;i<n;i++) if(pid[i]==id) idx=i;
                type=3;
            }
        }

        if(idx != -1){
            // Execution
            remTime[idx]--;
            currTime++;
            if(type==1 || type==2) slice[idx]++;

            // ADD NEW ARRIVALS (Critical for P2 before P1 logic)
            for(int i=0; i<n; i++){
                if(at[i] == currTime && remTime[i]>0 && prio[i]!=3 && !in_q[i]){
                    if(prio[i]==1) { q0.push(i); in_q[i]=true; }
                    else if(prio[i]==2) { q1.push(i); in_q[i]=true; }
                }
            }

            if(remTime[idx] == 0){
                completed++;
                ct[idx] = currTime;
                if(type==1) q0.pop();
                else if(type==2) q1.pop();
                // Q2 handled by getPID
            }
            else {
                // Check Quantum or Preemption
                bool drop = false;
                if(type==1 && slice[idx]==4) drop=true;
                if(type==2 && slice[idx]==8) drop=true;

                // Preemption Logic: If I am Q1, and Q0 has something now
                if(type==2 && !q0.empty()) drop=true;
                // Note: If preempted, slice is NOT reset in strict RR, but simplified here we often
                // If preempted, we should reset slice if we want "fresh" quantum next time, or keep
                // Standard MLQ: Preemption puts you back at tail.

                if(drop){
                    if(type==1) { q0.pop(); q0.push(idx); slice[idx]=0; }
                    else if(type==2) { q1.pop(); q1.push(idx); slice[idx]=0; }
                }
```

```
            }
        } else {
            currTime++;
            for(int i=0; i<n; i++){
                if(at[i] == currTime && remTime[i]>0 && prio[i]!=3 && !in_q[i]){
                    if(prio[i]==1) { q0.push(i); in_q[i]=true; }
                    else if(prio[i]==2) { q1.push(i); in_q[i]=true; }
                }
            }
        }
    }

    // Output
    cout<<"PID\tPrio\tAT\tBT\tCT\tTAT\tWT\tPR"<<endl;
    int sumtat=0, sumwt=0; float sumpr=0;
    for(int i=0;i<n;i++){
        tat[i]=ct[i]-at[i]; wt[i]=tat[i]-bt[i]; pr[i]=(float)tat[i]/bt[i];
        cout<<pid[i]<<"\t"<<prio[i]<<"\t"<<at[i]<<"\t"<<bt[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i
        sumtat+=tat[i]; sumwt+=wt[i]; sumpr+=pr[i];
    }
    cout<<"Avg TAT: "<<(float)sumtat/n<<endl;
    cout<<"Avg WT: "<<(float)sumwt/n<<endl;
    cout<<"Avg PR: "<<sumpr/n<<endl;
}
```

```
--- MLQ SCHEDULER ---
Enter number of processes: 5
AT P1: 0
AT P2: 1
AT P3: 2
AT P4: 3
AT P5: 4
BT P1: 5
BT P2: 4
BT P3: 3
BT P4: 2
BT P5: 1
Prio P1 (1=Sys, 2=Int, 3=Bat): 3
Prio P2 (1=Sys, 2=Int, 3=Bat): 1
Prio P3 (1=Sys, 2=Int, 3=Bat): 2
Prio P4 (1=Sys, 2=Int, 3=Bat): 1
Prio P5 (1=Sys, 2=Int, 3=Bat): 2
PID     Prio    AT      BT      CT      TAT     WT      PR
1       3       0       5       15      15      10      3
2       1       1       4       5       4       0       1
3       2       2       3       10      8       5       2.66667
4       1       3       2       7       4       2       2
5       2       4       1       11      7       6       7
Avg TAT: 7.6
Avg WT: 4.6
Avg PR: 3.13333
```

FIGURE 1. MLQ example

Question 2. Write a CPP program to simulate the CPU scheduling algorithm MLFQ(Multilevel Feedback Queue).

```cpp
#include<bits/stdc++.h>
using namespace std;

void bubbleSort(vector<int>&at, vector<int>&bt, vector<int>&pid){
    int n=at.size();
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-1;j++){
            if(at[j]>at[j+1] || (at[j]==at[j+1] && pid[j]>pid[j+1])){
                swap(at[j],at[j+1]); swap(pid[j],pid[j+1]); swap(bt[j],bt[j+1]);
            }
        }
    }
}

// SRTF Logic checks if process is in Q2 (Level 2)
int getPID(vector<int>&bt, vector<int>&at, vector<int>&remProcess, int currTime, vector<int>&remTime
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;
    for(int pid : remProcess){
        int i = pid - 1;
```

```
        if(at[i]<=currTime && remTime[i]>0 && level[i] == 2){
             pq.push({remTime[i], at[i]*1000 + pid});
        }
    }
    if(pq.empty()) return -1;
    return pq.top().second % 1000;
}

int main(){
    cout<<"--- MLFQ SCHEDULER ---"<<endl;
    int n; cout<<"Enter number of processes: "; cin>>n;
    vector<int>pid(n), at(n), bt(n), ct(n), tat(n), wt(n);
    vector<float>pr(n);
    for(int i=0;i<n;i++){ cout<<"AT P"<<i+1<<": "; cin>>at[i]; pid[i]=i+1; }
    for(int i=0;i<n;i++){ cout<<"BT P"<<i+1<<": "; cin>>bt[i]; }

    bubbleSort(at, bt, pid);

    vector<int>remTime(bt);
    vector<int>remProcess; for(int x:pid) remProcess.push_back(x);
    vector<int>slice(n,0);
    vector<int>level(n,0); // 0=Q0, 1=Q1, 2=Q2
    vector<bool>in_q(n, false);

    queue<int> q0, q1;
    int completed=0, currTime=at[0];

    // Initial load (All go to Q0)
    for(int i=0; i<n; i++){
        if(at[i]<=currTime){ q0.push(i); in_q[i]=true; }
    }

    while(completed<n){
        int idx = -1, type = 0;

        if(!q0.empty()){ idx=q0.front(); type=0; }
        else if(!q1.empty()){ idx=q1.front(); type=1; }
        else {
            int id = getPID(bt, at, remProcess, currTime, remTime, level);
            if(id!=-1) { for(int i=0;i<n;i++) if(pid[i]==id) idx=i; type=2; }
        }

        if(idx != -1){
            remTime[idx]--;
            currTime++;
            if(type==0 || type==1) slice[idx]++;

            // ADD NEW ARRIVALS to Q0
            for(int i=0; i<n; i++){
                if(at[i] == currTime && remTime[i]>0 && !in_q[i]){
                    q0.push(i); in_q[i]=true;
                }
            }

            if(remTime[idx] == 0){
                completed++;
```

```
                ct[idx] = currTime;
                if(type==0) q0.pop();
                else if(type==1) q1.pop();
            }
            else {
                // Logic: Pop, Demote, Push
                bool demote = false;
                if(type==0 && slice[idx]==4) {
                    q0.pop();
                    level[idx]=1; slice[idx]=0;
                    q1.push(idx); // Move to Q1
                }
                else if(type==1 && slice[idx]==8) {
                    q1.pop();
                    level[idx]=2; // Move to Q2 (SRTF)
                    // No push needed, getPID checks 'level' vector
                }
                else if(type==1 && !q0.empty()){
                    // Preemption: Q1 running, new Q0 process arrived
                    q1.pop();
                    q1.push(idx); // Put back in Q1
                    // Note: In strict MLFQ, we might keep slice or reset. Keeping usage is fair.
                }
            }
        } else {
            currTime++;
            for(int i=0; i<n; i++){
                if(at[i] == currTime && remTime[i]>0 && !in_q[i]){
                    q0.push(i); in_q[i]=true;
                }
            }
        }
    }

    cout<<"PID\tAT\tBT\tCT\tTAT\tWT\tPR"<<endl;
    int sumtat=0, sumwt=0; float sumpr=0;
    for(int i=0;i<n;i++){
        tat[i]=ct[i]-at[i]; wt[i]=tat[i]-bt[i]; pr[i]=(float)tat[i]/bt[i];
        cout<<pid[i]<<"\t"<<at[i]<<"\t"<<bt[i]<<"\t"<<ct[i]<<"\t"<<tat[i]<<"\t"<<wt[i]<<"\t"<<pr[i]<
        sumtat+=tat[i]; sumwt+=wt[i]; sumpr+=pr[i];
    }
    cout<<"Avg TAT: "<<(float)sumtat/n<<endl;
    cout<<"Avg WT: "<<(float)sumwt/n<<endl;
    cout<<"Avg PR: "<<sumpr/n<<endl;
}
```

```
--- MLFQ SCHEDULER ---
Enter number of processes: 5
AT P1: 0
AT P2: 2
AT P3: 3
AT P4: 4
AT P5: 5
BT P1: 15
BT P2: 13
BT P3: 19
BT P4: 21
BT P5: 5
PID     AT      BT      CT      TAT     WT      PR
1       0       15      57      57      42      3.8
2       2       13      54      52      39      4
3       3       19      64      61      42      3.21053
4       4       21      73      69      48      3.28571
5       5       5       53      48      43      9.6
Avg TAT: 57.4
Avg WT: 42.8
Avg PR: 4.77925
```

FIGURE 2. MLFQ example