VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CSE2005 OS – LAB 01

Advait Deochakke
20BCE1143

Priority (non-primitive), Priority (Primitive), RoundRobin

→ Priority Scheduling (Non-Primitive)

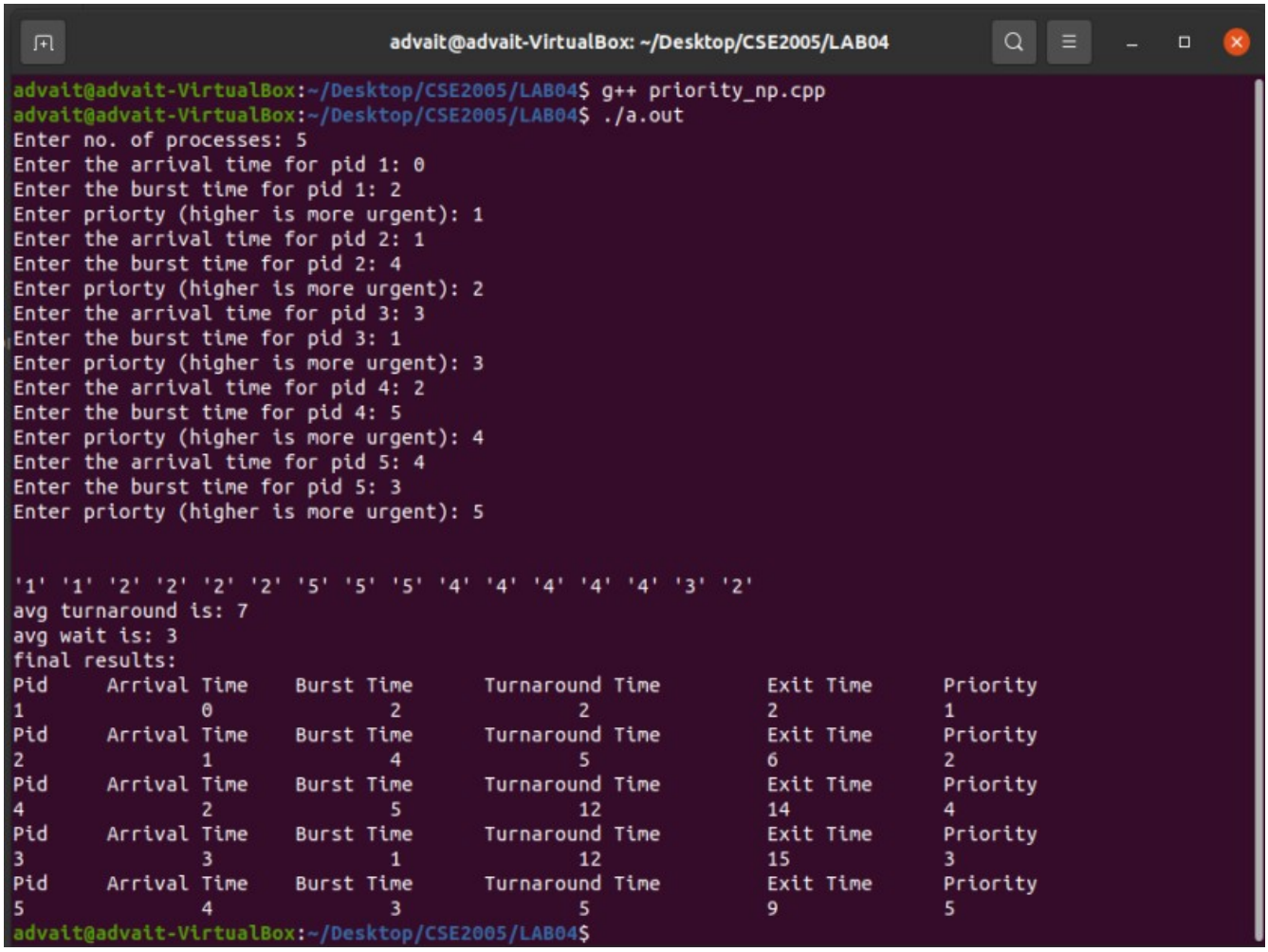
```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

struct process
{
  int pid;
  int arrtime;
  int burstime;
  int remaintime;
  int priority; //higher is more urgent
  int wtime=999;
  int turnaroundtime;
  int exitime;

};

bool compareArr(process p1, process p2)
{
  return(p1.arrtime > p2.arrtime);
}

bool compareRem(process p1, process p2)
{
```

```
  return(p1.remaintime > p2.remaintime);
}

bool comparePri(process p1, process p2)
{
  return(p1.priority < p2.priority);
}

int main()
{
  int n;
  int avgturnaround=0, avgwait=0;
  cout<<"Enter no. of processes: ";
  cin>>n;
  int curtime=0, ttlexec=0;
  vector<process> notinqueue, waitqueue, inqueue, completed;

  for(int i=1; i<n+1; i++)
  {
    process a;
    a.pid=i;
    cout<<"Enter the arrival time for pid "<<i<<": ";
    cin>>a.arrtime;
    cout<<"Enter the burst time for pid "<<i<<": ";
    cin>>a.burstime;
    a.remaintime=a.burstime;
    cout<<"Enter priorty (higher is more urgent): ";
    cin>>a.priority;
    ttlexec=ttlexec+a.burstime;

    notinqueue.push_back(a);

  }
  cout<<endl<<endl;

  sort(notinqueue.begin(), notinqueue.end(), compareArr);

  for(curtime; curtime<=ttlexec; curtime++)
  {
    int k1=999;
    if(!notinqueue.empty())
      k1=notinqueue.back().arrtime;

    if(inqueue.empty())
```

```
    {
      //only takes one process per time cause im lazy sry, 
could use like while loop and update k1 inside
      if(k1==curtime)
      {
        process a=notinqueue.back();
        notinqueue.pop_back();
        inqueue.push_back(a);
        inqueue.back().remaintime--;
        cout<<"'"<<inqueue.back().pid<<"' ";
        inqueue.back().wtime=0;
        avgwait=avgwait+curtime;
        if(inqueue.back().remaintime==0)
        {
          inqueue.back().exitime=curtime+1;
          inqueue.back().turnaroundtime=inqueue.back().exitime-
inqueue.back().arrtime;
          
avgturnaround=avgturnaround+inqueue.back().turnaroundtime;
          completed.push_back(inqueue.back());
          inqueue.pop_back();
          
        }
      }
    }
    else
    {
      if(k1==curtime)
      {
        process a=notinqueue.back();
        notinqueue.pop_back();
        waitqueue.push_back(a);
      }

      if(!inqueue.empty())
      {
        inqueue.back().remaintime--;
        cout<<"'"<<inqueue.back().pid<<"' ";
      }

      if(inqueue.back().wtime!=999)
      {
        inqueue.back().wtime=curtime;
```

```
        avgwait=avgwait+curtime;
      }

      if(inqueue.back().remaintime==0)
      {
        inqueue.back().exitime=curtime+1;
        inqueue.back().turnaroundtime=inqueue.back().exitime-
inqueue.back().arrtime;

avgturnaround=avgturnaround+inqueue.back().turnaroundtime;
        process a=inqueue.back();
        completed.push_back(a);
        inqueue.pop_back();
        sort(waitqueue.begin(), waitqueue.end(), comparePri);//
for non-primitive
        inqueue.push_back(waitqueue.back());
        waitqueue.pop_back();
      }
    }
  }

  avgturnaround=avgturnaround/n;
  avgwait=avgwait/n;

  cout<<"\navg turnaround is: "<<avgturnaround;
  cout<<"\navg wait is: "<<avgwait<<endl;

  cout<<"final results: "<<endl;

  sort(completed.begin(), completed.end(), compareArr);

  for(int i=0; i<n; i++)
  {
    cout<<"Pid\tArrival Time\tBurst Time\tTurnaround Time\t\
tExit Time\tPriority\n";
    cout<<completed.back().pid<<"\t\
t"<<completed.back().arrtime<<"\t\t"<<
    completed.back().burstime<<"\t\
t"<<completed.back().turnaroundtime<<"\t\t"<<
    completed.back().exitime<<"\t\
t"<<completed.back().priority<<"\n";
    completed.pop_back();
  }
  return 0;
```

```
}
```


advait@advait-VirtualBox: ~/Desktop/CSE2005/LAB04
advait@advait-VirtualBox:~/Desktop/CSE2005/LAB04$ g++ priority_np.cpp
advait@advait-VirtualBox:~/Desktop/CSE2005/LAB04$ ./a.out
Enter no. of processes: 5
Enter the arrival time for pid 1: 0
Enter the burst time for pid 1: 2
Enter priorty (higher is more urgent): 1
Enter the arrival time for pid 2: 1
Enter the burst time for pid 2: 4
Enter priorty (higher is more urgent): 2
Enter the arrival time for pid 3: 3
Enter the burst time for pid 3: 1
Enter priorty (higher is more urgent): 3
Enter the arrival time for pid 4: 2
Enter the burst time for pid 4: 5
Enter priorty (higher is more urgent): 4
Enter the arrival time for pid 5: 4
Enter the burst time for pid 5: 3
Enter priorty (higher is more urgent): 5
'1' '1' '2' '2' '2' '2' '5' '5' '5' '4' '4' '4' '4' '4' '3' '2'
avg turnaround is: 7
avg wait is: 3
final results:
Pid Arrival Time Burst Time Turnaround Time Exit Time Priority
1 0 2 2 2 1
Pid Arrival Time Burst Time Turnaround Time Exit Time Priority
2 1 4 5 6 2
Pid Arrival Time Burst Time Turnaround Time Exit Time Priority
4 2 5 12 14 4
Pid Arrival Time Burst Time Turnaround Time Exit Time Priority
3 3 1 12 15 3
Pid Arrival Time Burst Time Turnaround Time Exit Time Priority
5 4 3 5 9 5
advait@advait-VirtualBox:~/Desktop/CSE2005/LAB04$

<next page>

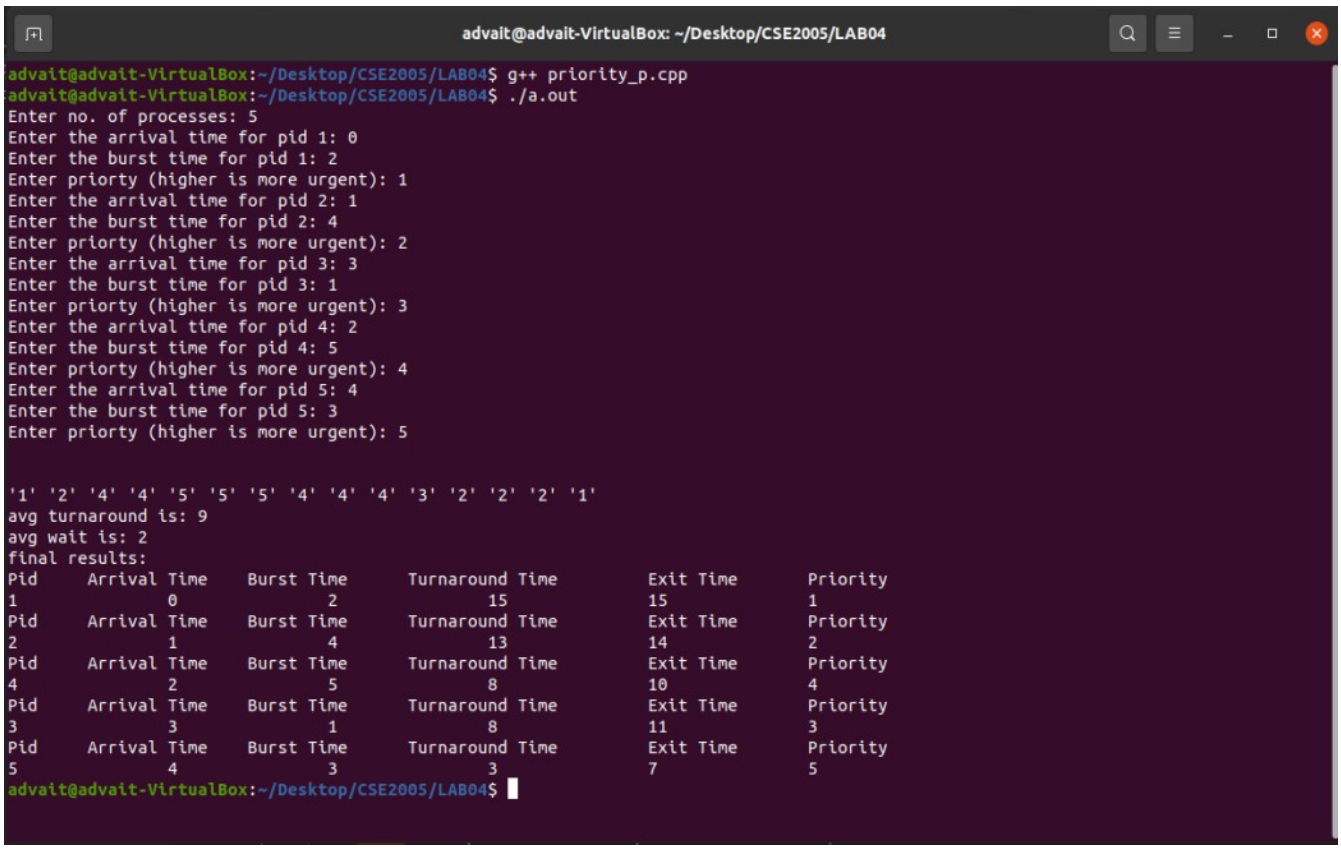
→ Priority Scheduling (Primitive)

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

struct process
{
  int pid;
  int arrtime;
  int burstime;
  int remaintime;
  int priority; //higher is more urgent
  int wtime=999;
  int turnaroundtime;
  int exitime;
};

bool compareArr(process p1, process p2)
{
  return(p1.arrtime > p2.arrtime);
}

bool compareRem(process p1, process p2)
{
  return(p1.remaintime > p2.remaintime);
}

bool comparePri(process p1, process p2)
{
  return(p1.priority < p2.priority);
}

int main()
{
  int n;
  int avgturnaround=0, avgwait=0;
  cout<<"Enter no. of processes: ";
  cin>>n;
  int curtime=0, ttlexec=0;
  vector<process> notinqueue, inqueue, completed;
```

```
  for(int i=1; i<n+1; i++)
  {
    process a;
    a.pid=i;
    cout<<"Enter the arrival time for pid "<<i<<": ";
    cin>>a.arrtime;
    cout<<"Enter the burst time for pid "<<i<<": ";
    cin>>a.burstime;
    a.remaintime=a.burstime;
    cout<<"Enter priorty (higher is more urgent): ";
    cin>>a.priority;
    ttlexec=ttlexec+a.burstime;

    notinqueue.push_back(a);

  }
  cout<<endl<<endl;

  sort(notinqueue.begin(), notinqueue.end(), compareArr);

  for(curtime; curtime<=ttlexec; curtime++)
  {
    int k1=999;
    if(!notinqueue.empty())
      k1=notinqueue.back().arrtime;

    if(inqueue.empty())
    {
      //only takes one process per time cause im lazy sry,
could use like while loop and update k1 inside
      if(k1==curtime)
      {
        process a=notinqueue.back();
        notinqueue.pop_back();
        inqueue.push_back(a);
        inqueue.back().remaintime--;
        cout<<"'"<<inqueue.back().pid<<"' ";
        inqueue.back().wtime=0;
        avgwait=avgwait+curtime;
        if(inqueue.back().remaintime==0)
        {
          inqueue.back().exitime=curtime+1;
```

```
          inqueue.back().turnaroundtime=inqueue.back().exitime-
inqueue.back().arrtime;

avgturnaround=avgturnaround+inqueue.back().turnaroundtime;
          process a=inqueue.back();
          completed.push_back(a);
          inqueue.pop_back();

        }
      }
    }
    else
    {
      if(k1==curtime)
      {
        process a=notinqueue.back();
        notinqueue.pop_back();
        inqueue.push_back(a);
      }

      sort(inqueue.begin(), inqueue.end(), comparePri); //for
primitive

      if(!inqueue.empty())
      {
        inqueue.back().remaintime--;
        cout<<"'"<<inqueue.back().pid<<"' ";
      }
      if(inqueue.back().wtime!=999)
      {
        inqueue.back().wtime=curtime;
        avgwait=avgwait+curtime;
      }
      if(inqueue.back().remaintime==0)
      {
        inqueue.back().exitime=curtime+1;
        inqueue.back().turnaroundtime=inqueue.back().exitime-
inqueue.back().arrtime;

avgturnaround=avgturnaround+inqueue.back().turnaroundtime;
        process a=inqueue.back();
        completed.push_back(a);
        inqueue.pop_back();
```

```
        sort(inqueue.begin(), inqueue.end(), comparePri); //for
primitive
      }
    }
  }

  avgturnaround=avgturnaround/n;
  avgwait=avgwait/n;

  cout<<"\navg turnaround is: "<<avgturnaround;
  cout<<"\navg wait is: "<<avgwait<<endl;

  cout<<"final results: "<<endl;

  sort(completed.begin(), completed.end(), compareArr);

  for(int i=0; i<n; i++)
  {
    cout<<"Pid\tArrival Time\tBurst Time\tTurnaround Time\t\
tExit Time\tPriority\n";

    cout<<completed.back().pid<<"\t\
t"<<completed.back().arrtime<<"\t\t"<<
    completed.back().burstime<<"\t\
t"<<completed.back().turnaroundtime<<"\t\t"<<
    completed.back().exitime<<"\t\
t"<<completed.back().priority<<"\n";

    completed.pop_back();
  }

  return 0;
}
```



→ RoudRobin (Primitive by default)

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

struct process
{
  int pid;
  int arrtime;
  int burstime;
  int remaintime;
  int wtime=999;
  int turnaroundtime;
  int exitime;
};

bool compareArr(process p1, process p2)
{
  return(p1.arrtime > p2.arrtime);
}
```

```
bool compareRem(process p1, process p2)
{
  return(p1.remaintime > p2.remaintime);
}

int main()
{
  int n;
  int avgturnaround=0, avgwait=0;
  int keeper=0;
  cout<<"Enter no. of processes: ";
  cin>>n;
  int curtime=0, ttlexec=0;
  vector<process> notinqueue, inqueue, robined, completed;

  for(int i=1; i<n+1; i++)
  {
    process a;
    a.pid=i;
    cout<<"Enter the arrival time for pid "<<i<<": ";
    cin>>a.arrtime;
    cout<<"Enter the burst time for pid "<<i<<": ";
    cin>>a.burstime;
    a.remaintime=a.burstime;
    //cout<<"Enter priorty (higher is more urgent): ";
    //cin>>a.priority;
    ttlexec=ttlexec+a.burstime;
    notinqueue.push_back(a);
  }
  cout<<endl<<endl;

  sort(notinqueue.begin(), notinqueue.end(), compareArr);

  for(curtime; curtime<=ttlexec; curtime++)
  {
    keeper++;

    int k1=999;
    if(!notinqueue.empty())
      k1=notinqueue.back().arrtime;

    if(inqueue.empty())
```

```
    {
      //only takes one process per time cause im lazy sry, could use like while loop and update k1 inside
      if(k1==curtime)
      {
        process a=notinqueue.back();
        notinqueue.pop_back();
        inqueue.push_back(a);
        inqueue.back().remaintime--;
        cout<<"'"<<inqueue.back().pid<<"' ";
        inqueue.back().wtime=0;
        avgwait=avgwait+curtime;
        if(inqueue.back().remaintime==0)
        {
          inqueue.back().exitime=curtime+1;
          inqueue.back().turnaroundtime=inqueue.back().exitime-inqueue.back().arrtime;
          avgturnaround=avgturnaround+inqueue.back().turnaroundtime;
          process a=inqueue.back();
          completed.push_back(a);
          inqueue.pop_back();
        }
      }
    }
    else
    {
      if(k1==curtime)
      {
        process a=notinqueue.back();
        notinqueue.pop_back();
        inqueue.push_back(a);
      }

      sort(inqueue.begin(), inqueue.end(), compareArr); //for primitive

      if(!inqueue.empty())
      {
        inqueue.back().remaintime--;
        cout<<"'"<<inqueue.back().pid<<"' ";
      }
```

```
      if(inqueue.back().wtime!=999)
      {
        inqueue.back().wtime=curtime;
        avgwait=avgwait+curtime;
      }

      if(inqueue.back().remaintime==0)
      {
        inqueue.back().exitime=curtime+1;
        inqueue.back().turnaroundtime=inqueue.back().exitime-
inqueue.back().arrtime;

avgturnaround=avgturnaround+inqueue.back().turnaroundtime;
        process a=inqueue.back();
        completed.push_back(a);
        inqueue.pop_back();
        keeper=0;
      }
      else if(keeper==2)
      {
        robined.push_back(inqueue.back());
        inqueue.pop_back();
        keeper=0;
      }

      if(inqueue.empty())
      {
        int test1=robined.size();
        for(int i5=0; i5<test1; i5++)
        {
          inqueue.push_back(robined.back());
          robined.pop_back();
        }
      }
    }
  }

  avgturnaround=avgturnaround/n;
  avgwait=avgwait/n;

  cout<<"\navg turnaround is: "<<avgturnaround;
  cout<<"\navg wait is: "<<avgwait<<endl;
```

```
  cout<<"final results: "<<endl;

  sort(completed.begin(), completed.end(), compareArr);

  for(int i=0; i<n; i++)
  {
    cout<<"Pid\tArrival Time\tBurst Time\tTurnaround Time\t\
tExit Time\n";

    cout<<completed.back().pid<<"\t\
t"<<completed.back().arrtime<<"\t\t"<<
    completed.back().burstime<<"\t\
t"<<completed.back().turnaroundtime<<"\t\t"<<
    completed.back().exitime<<"\n";

    completed.pop_back();
  }

  return 0;
}
```

```
advait@advait-VirtualBox: ~/Desktop/CSE2005/LAB04
advait@advait-VirtualBox:~/Desktop/CSE2005/LAB04$ g++ roundrobin.cpp
advait@advait-VirtualBox:~/Desktop/CSE2005/LAB04$ ./a.out
Enter no. of processes: 5
Enter the arrival time for pid 1: 0
Enter the burst time for pid 1: 2
Enter the arrival time for pid 2: 1
Enter the burst time for pid 2: 4
Enter the arrival time for pid 3: 3
Enter the burst time for pid 3: 1
Enter the arrival time for pid 4: 2
Enter the burst time for pid 4: 5
Enter the arrival time for pid 5: 4
Enter the burst time for pid 5: 3


'1' '1' '2' '2' '4' '4' '3' '5' '5' '2' '2' '4' '4' '5' '4'
avg turnaround is: 7
avg wait is: 0
final results:
Pid     Arrival Time    Burst Time      Turnaround Time         Exit Time
1               0               2               2               2
Pid     Arrival Time    Burst Time      Turnaround Time         Exit Time
2               1               4               10              11
Pid     Arrival Time    Burst Time      Turnaround Time         Exit Time
4               2               5               13              15
Pid     Arrival Time    Burst Time      Turnaround Time         Exit Time
3               3               1               4               7
Pid     Arrival Time    Burst Time      Turnaround Time         Exit Time
5               4               3               10              14
advait@advait-VirtualBox:~/Desktop/CSE2005/LAB04$
```