Process scheduler

3rd CSE Operating-System Structures

Fady Fares Abdel Ahad Sobhy

Sec: 2

Contents

[2 How to use the scheduler 2](#_Toc419917478)

[2.1 Open the processes scheduler.exe 2](#_Toc419917479)

[2.2 choose one of the available schedulers and determine the number of processes and the quantum (in case of Round robin) then press on edit button 2](#_Toc419917480)

[2.3 new window will be opened to edit the processes, fill the required fields then press done. 3](#_Toc419917481)

[2.4 now you can press simulate to see the scheduled processes in to forms 4](#_Toc419917482)

[3 Restrictions and assumptions 5](#_Toc419917483)

[4 Test cases 7](#_Toc419917484)

[4.1 FCFS 7](#_Toc419917485)

[4.2 SJF Non-Preemptive 8](#_Toc419917486)

[4.3 SJF Preemptive 9](#_Toc419917487)

[4.4 Priority non-preemptive 10](#_Toc419917488)

[4.5 Priority preemptive 11](#_Toc419917489)

[4.6 Round Robin 12](#_Toc419917490)

[5 parts from the code 13](#_Toc419917491)

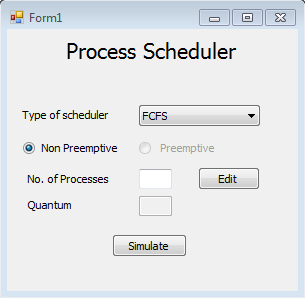
[5.1 FCFS 13](#_Toc419917492)

[5.2 SJF non preemptive 13](#_Toc419917493)

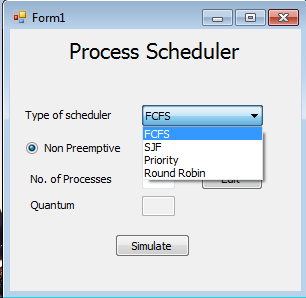
[5.3 SJF Preemptive 15](#_Toc419917494)

# How to use the scheduler

## Open the processes scheduler.exe



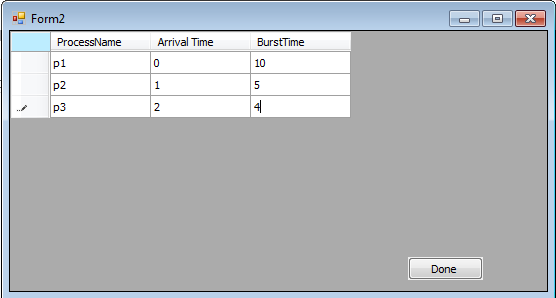
## choose one of the available schedulers and determine the number of processes and the quantum (in case of Round robin) then press on edit button



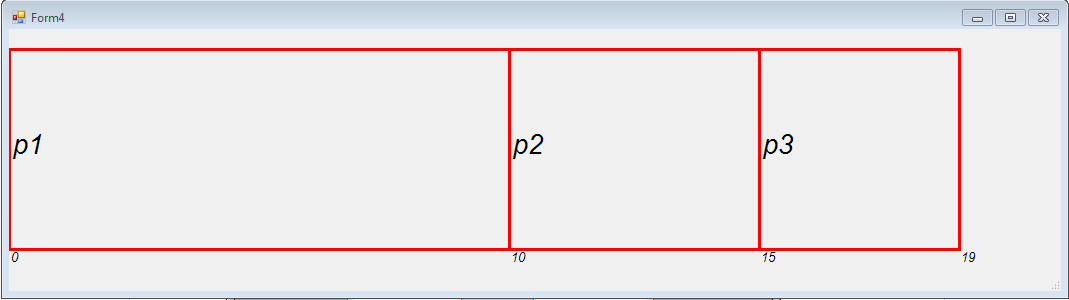
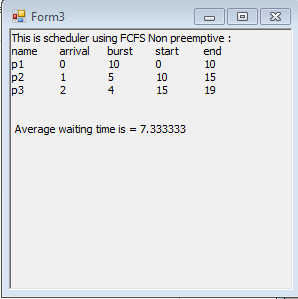
* supported schedulers are :

1. FCFS
   1. Non-preemptive.
2. SJF
   1. Non-preemptive.
   2. Preemptive
3. Priority
   1. Non-preemptive.
   2. Preemptive
4. Round Robin
   1. Non-preemptive.

## new window will be opened to edit the processes, fill the required fields then press done.

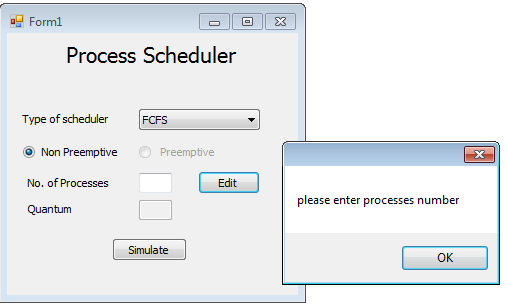


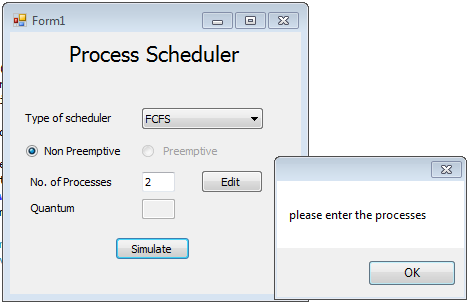
## now you can press simulate to see the scheduled processes in to forms

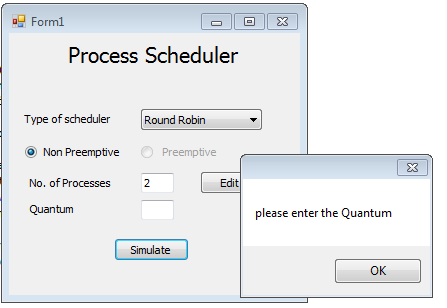


# Restrictions and assumptions

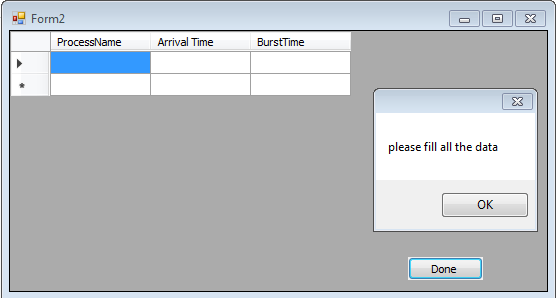
1. Input values must be in integers and it’s not allowed to enter any different types.
2. It’s not allowed to start simulation without putting processes no value and editing the processes and choosing the quantum (in case of Round Robin).







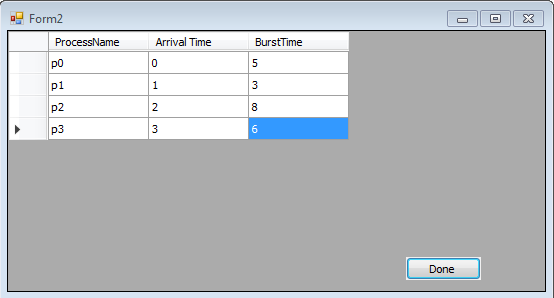
1. It’s not allowed to let any empty field in the edit window and it’s prepared by the number of processes which has been already defined and it’s not allowed to enter any extra fields.

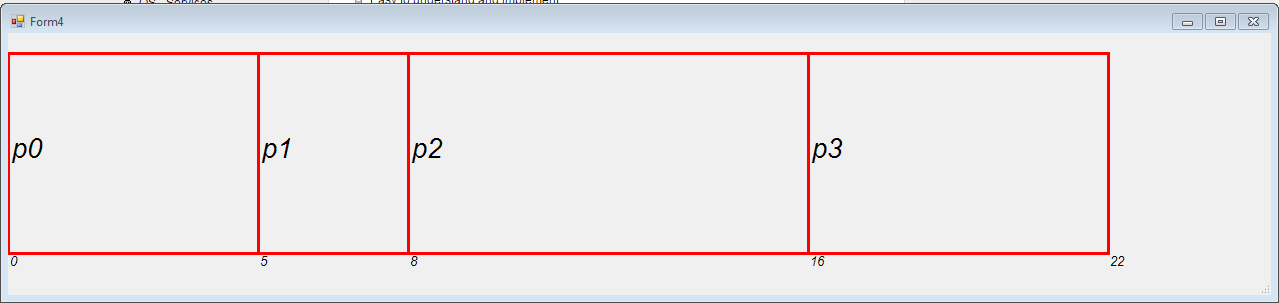
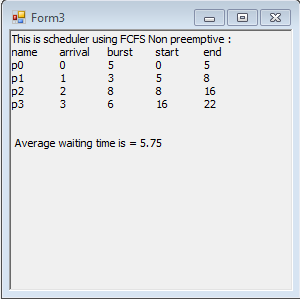


1. It’s not allowed to write in the simulation windows
2. The graph window is not fully informed and not so accurate so please take the scheduler table into consideration.

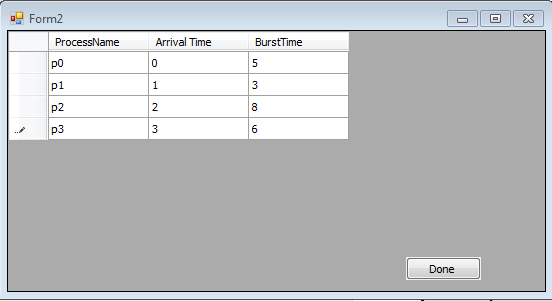
# Test cases

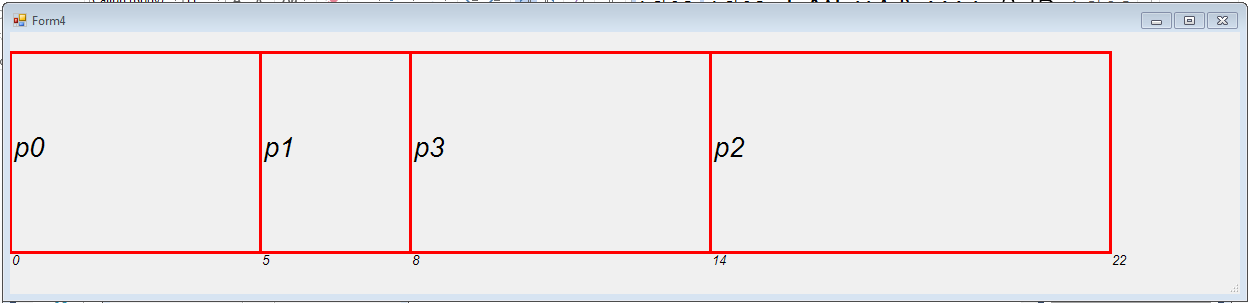
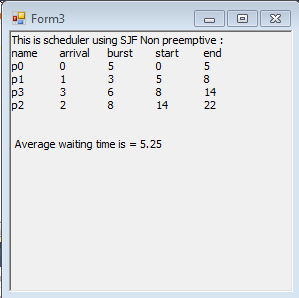
## FCFS



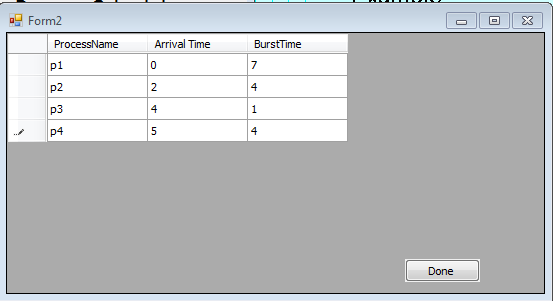


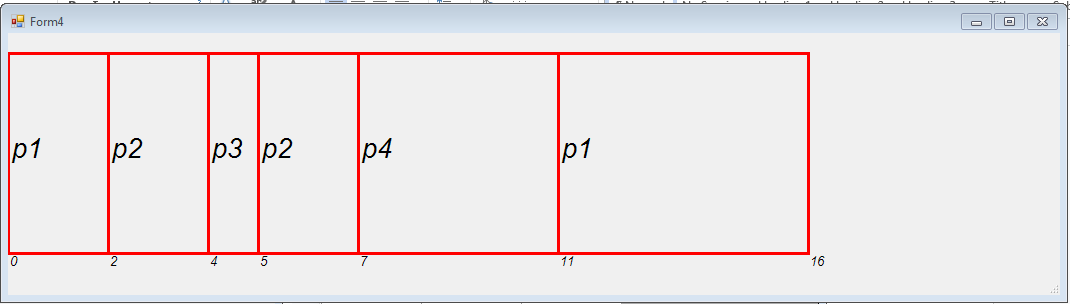
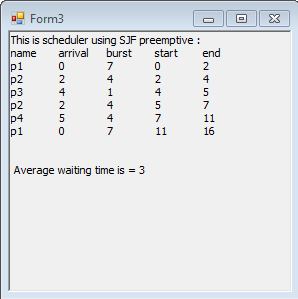
## SJF Non-Preemptive



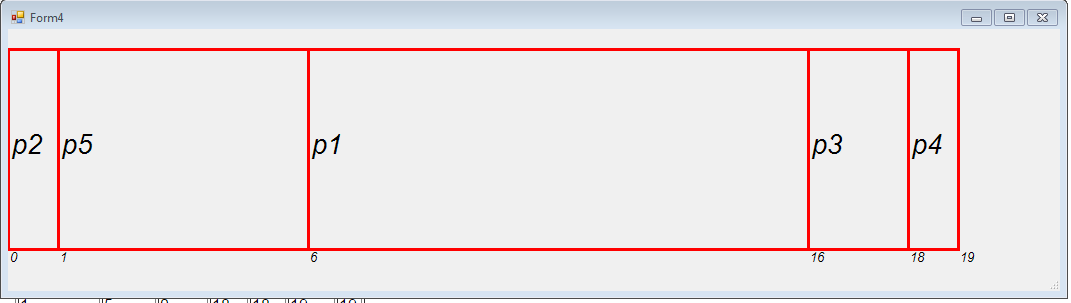
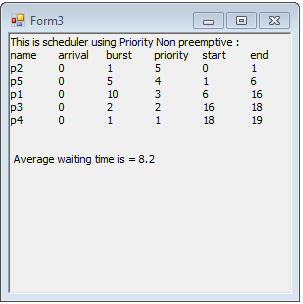
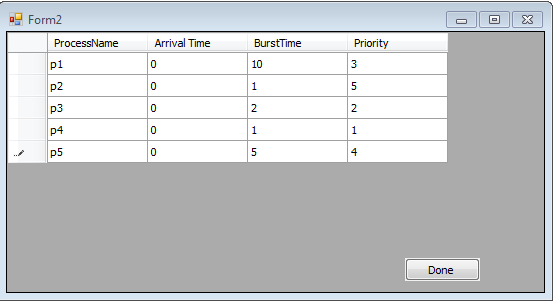


## SJF Preemptive

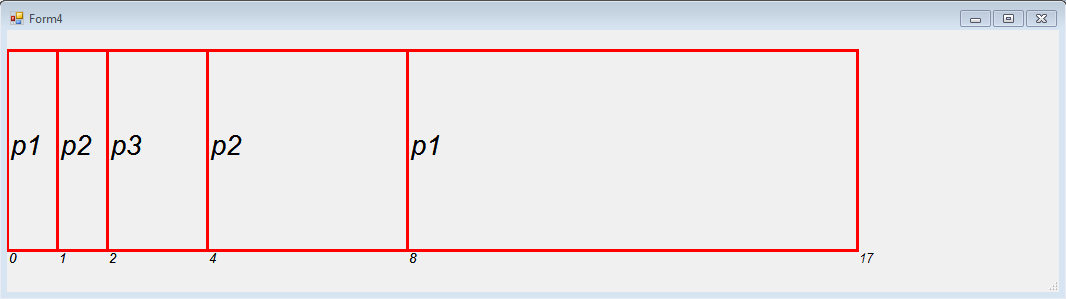
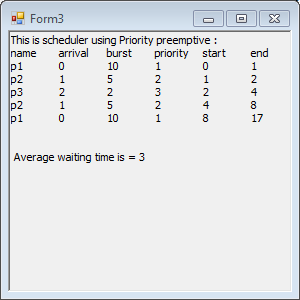
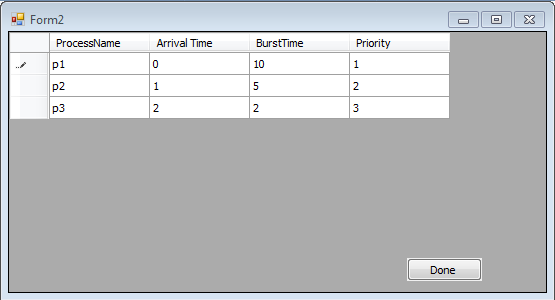




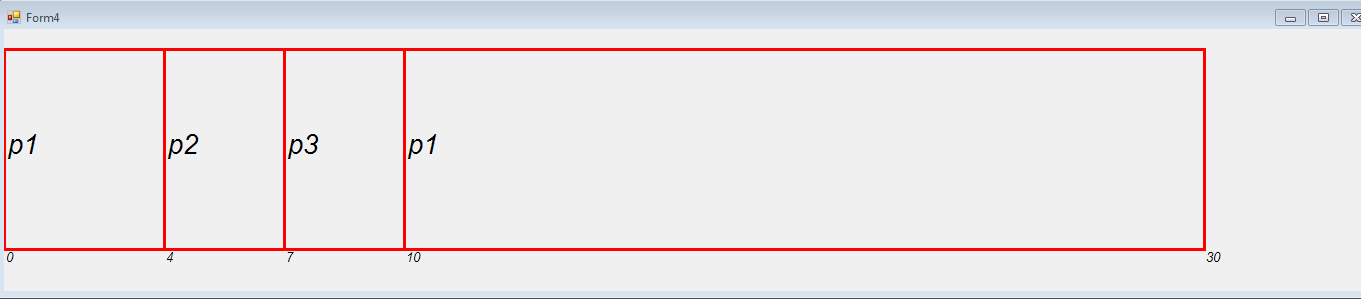
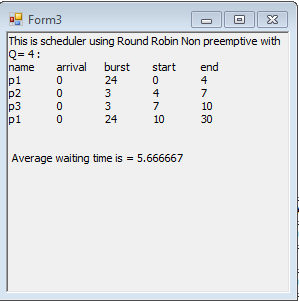
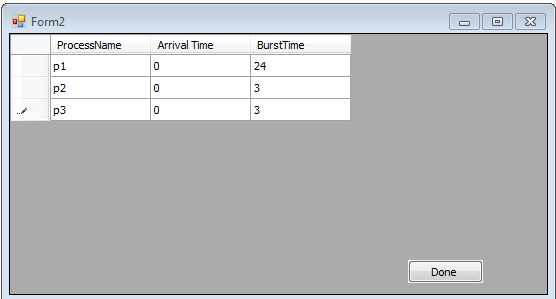
## Priority non-preemptive



## Priority preemptive



## Round Robin



# parts from the code

## FCFS

public void FCFS()

{

Avergtime = 0;

linked = new LinkedList<process>(linked.OrderBy(item => item.arrive));

linked.ElementAt(0).start = linked.ElementAt(0).arrive;

linked.ElementAt(0).end = linked.ElementAt(0).arrive + linked.ElementAt(0).burst;

for (int i = 1; i < linked.Count; i++)

{

linked.ElementAt(i).start = linked.ElementAt(i - 1).end;

linked.ElementAt(i).end = linked.ElementAt(i - 1).end + linked.ElementAt(i).burst;

}

for (int i = 0; i < linked.Count; i++)

{

linked.ElementAt(i).wait = linked.ElementAt(i).start - linked.ElementAt(i).arrive;

Avergtime += linked.ElementAt(i).wait;

}

Avergtime = Avergtime / linked.Count;

}

## SJF non preemptive

public void SJF()

{

Avergtime = 0;

linked = new LinkedList<process>(linked.OrderBy(item => item.arrive));

LinkedList<process> larrive = new LinkedList<process>();

linked.ElementAt(0).start = linked.ElementAt(0).arrive;

linked.ElementAt(0).end = linked.ElementAt(0).arrive + linked.ElementAt(0).burst;

process p1 = linked.ElementAt(0);

larrive.AddLast(p1);

linked.Remove(linked.ElementAt(0));

while (p1.arrive == linked.ElementAt(0).arrive)

{

larrive.AddLast(linked.ElementAt(0));

linked.Remove(linked.ElementAt(0));

if (linked.Count == 0)

break;

}

larrive = new LinkedList<process>(larrive.OrderBy(item => item.burst));

larrive.ElementAt(0).start = larrive.ElementAt(0).arrive;

larrive.ElementAt(0).end = larrive.ElementAt(0).arrive + larrive.ElementAt(0).burst;

for (int i = 1; i < larrive.Count; i++)

{

larrive.ElementAt(i).start = larrive.ElementAt(i - 1).end;

larrive.ElementAt(i).end = larrive.ElementAt(i - 1).end + larrive.ElementAt(i).burst;

}

if (linked.Count != 0)

{

if (larrive.ElementAt(larrive.Count - 1).end >= linked.ElementAt(0).arrive)

{

linked = new LinkedList<process>(linked.OrderBy(item => item.burst));

linked.ElementAt(0).start = larrive.ElementAt(larrive.Count - 1).end;

linked.ElementAt(0).end = linked.ElementAt(0).start + linked.ElementAt(0).burst;

for (int i = 1; i < linked.Count; i++)

{

linked.ElementAt(i).start = linked.ElementAt(i - 1).end;

linked.ElementAt(i).end = linked.ElementAt(i - 1).end + linked.ElementAt(i).burst;

}

linked = new LinkedList<process>(larrive.Concat(linked));

}

else

{

linked = new LinkedList<process>(linked.OrderBy(item => item.arrive).ThenBy(item => item.burst));

linked.ElementAt(0).start = larrive.ElementAt(larrive.Count - 1).end;

linked.ElementAt(0).end = linked.ElementAt(0).start + linked.ElementAt(0).burst;

for (int i = 1; i < linked.Count; i++)

{

linked.ElementAt(i).start = linked.ElementAt(i - 1).end;

linked.ElementAt(i).end = linked.ElementAt(i - 1).end + linked.ElementAt(i).burst;

}

linked = new LinkedList<process>(larrive.Concat(linked));

}

}

else

{

linked = new LinkedList<process>(larrive.Concat(linked));

linked = new LinkedList<process>(linked.OrderBy(item => item.arrive).ThenByDescending(item => item.priority));

}

for (int i = 0; i < linked.Count; i++)

{

linked.ElementAt(i).wait = linked.ElementAt(i).start - linked.ElementAt(i).arrive;

Avergtime += linked.ElementAt(i).wait;

}

Avergtime = Avergtime / linked.Count;

}

## SJF Preemptive

public void SJF\_Prem()

{

Avergtime = 0;

linked = new LinkedList<process>(linked.OrderBy(item => item.arrive));

LinkedList<process> bigger = new LinkedList<process>();

LinkedList<process> waiting = new LinkedList<process>();

linked.ElementAt(0).start = linked.ElementAt(0).arrive;

process p1 = linked.ElementAt(0);

bigger.AddLast(p1);

linked.Remove(linked.ElementAt(0));

while (p1.arrive == linked.ElementAt(0).arrive)

{

bigger.AddLast(linked.ElementAt(0));

linked.Remove(linked.ElementAt(0));

if (linked.Count == 0)

break;

}

bigger = new LinkedList<process>(bigger.OrderBy(item => item.burst));

for (int i = 1; i < bigger.Count; i++)

{

waiting.AddLast(bigger.ElementAt(i));

bigger.Remove(bigger.ElementAt(i));

}

waiting = new LinkedList<process>(waiting.OrderBy(item => item.burst));

while (linked.Count != 0)

{

process p = new process();

p.name = linked.ElementAt(0).name;

p.arrive = linked.ElementAt(0).arrive;

p.burst = linked.ElementAt(0).burst;

p.start = linked.ElementAt(0).start;

p.end = linked.ElementAt(0).end;

waiting.AddLast(p);

linked.Remove(linked.ElementAt(0));

waiting = new LinkedList<process>(waiting.OrderBy(item => item.burst));

if ((bigger.ElementAt(bigger.Count - 1).burst - waiting.ElementAt(0).arrive > waiting.ElementAt(0).burst) || (bigger.ElementAt(bigger.Count - 1).burst > waiting.ElementAt(0).burst))

{

process t = new process();

t.name = bigger.ElementAt(bigger.Count - 1).name;

t.arrive = bigger.ElementAt(bigger.Count - 1).arrive;

t.burst = (bigger.ElementAt(bigger.Count - 1).burst - (waiting.ElementAt(0).arrive - (bigger.ElementAt(bigger.Count - 1).start)));

t.start = bigger.ElementAt(bigger.Count - 1).end;

t.end = waiting.ElementAt(0).arrive;

// if ((bigger.ElementAt(bigger.Count - 1).burst - waiting.ElementAt(0).arrive > waiting.ElementAt(0).burst))

bigger.ElementAt(bigger.Count - 1).end = waiting.ElementAt(0).arrive;

/\* else

bigger.ElementAt(bigger.Count - 1).end = bigger.ElementAt(bigger.Count - 1).start + bigger.ElementAt(bigger.Count - 1).burst;

\*/

waiting.AddLast(t);

process d = new process();

d.name = waiting.ElementAt(0).name;

d.arrive = waiting.ElementAt(0).arrive;

d.burst = waiting.ElementAt(0).burst;

d.start = bigger.ElementAt(bigger.Count - 1).end;

d.end = waiting.ElementAt(0).end;

waiting.ElementAt(0).start = bigger.ElementAt(bigger.Count - 1).end;

bigger.AddLast(d);

waiting.Remove(waiting.ElementAt(0));

waiting = new LinkedList<process>(waiting.OrderBy(item => item.burst));

if (waiting.ElementAt(0).burst < 0)

{

waiting.ElementAt(0).burst = 0;

waiting.Remove(waiting.ElementAt(0));

}

}

else

{

waiting.ElementAt(0).start = bigger.ElementAt(bigger.Count - 1).end;

waiting.ElementAt(0).end = waiting.ElementAt(0).start + waiting.ElementAt(0).burst;

for (int i = 1; i < waiting.Count; i++)

{

waiting.ElementAt(i).start = waiting.ElementAt(i - 1).end;

waiting.ElementAt(i).end = waiting.ElementAt(i - 1).end + waiting.ElementAt(i).burst;

}

}

bigger.ElementAt(bigger.Count - 1).end = bigger.ElementAt(bigger.Count - 1).start + bigger.ElementAt(bigger.Count - 1).burst;

}

waiting.ElementAt(0).start = bigger.ElementAt(bigger.Count - 1).end;

waiting.ElementAt(0).end = waiting.ElementAt(0).start + waiting.ElementAt(0).burst;

for (int i = 1; i < waiting.Count; i++)

{

waiting.ElementAt(i).start = waiting.ElementAt(i - 1).end;

waiting.ElementAt(i).end = waiting.ElementAt(i - 1).end + waiting.ElementAt(i).burst;

}

bigger = new LinkedList<process>(bigger.Concat(waiting));

linked = bigger;

LinkedList<process> wait = new LinkedList<process>();

wait = linked;

wait = new LinkedList<process>(linked.OrderBy(item => item.name).ThenBy((item => item.end)));

for (int i = 1; i < wait.Count; i++)

{

if (wait.ElementAt(i).name == wait.ElementAt(i - 1).name)

{

wait.ElementAt(i).burst = wait.ElementAt(i - 1).burst;

wait.Remove(wait.ElementAt(i - 1));

}

}

for (int i = 0; i < wait.Count; i++)

{

wait.ElementAt(i).wait = wait.ElementAt(i).end - wait.ElementAt(i).arrive - wait.ElementAt(i).burst;

Avergtime = Avergtime + wait.ElementAt(i).wait;

}

Avergtime = Avergtime / wait.Count;

}