DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

LAB 04 – Process Management

Process:

A process can be thought of as a program in execution. A process will need certain resources—such as CPU time, memory, files, and I/O devices to accomplish its task. These resources are allocated to the process either when it is created or while it is executing.

A process is the unit of work in most systems. Systems consist of a collection of processes: operating-system processes execute system code, and user processes execute user code. All these processes may execute concurrently.

A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables.A process may also include a heap,which is memory that is dynamically allocated during process run time. The structure of a process in memory is shown in Figure 4.1.

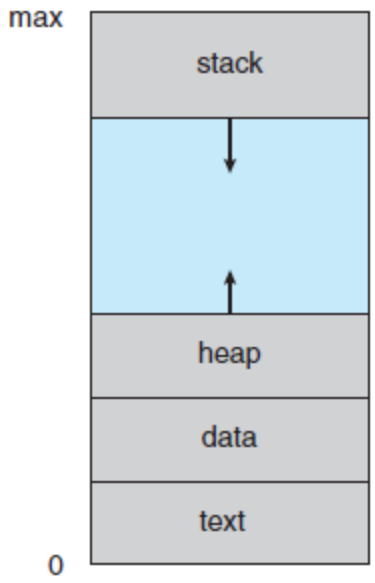

Figure 4.1 Process in memory.

Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

- **New:** The process is being created.
- **Running:** Instructions are being executed.
- **Waiting:** The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- **Ready:** The process is waiting to be assigned to a processor.
- **Terminated:** The process has finished execution.

Process Control Block

Each process is represented in the operating system by a process control block (PCB)—also called a task control block.A PCB is shown in Figure 4.2. It contains many pieces of information associated with a specific process.

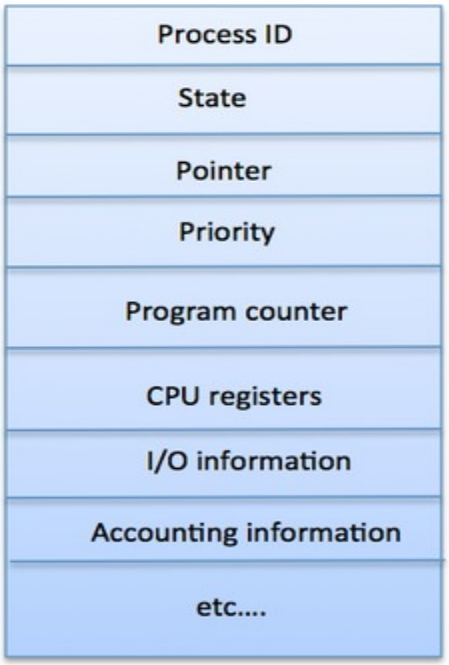

Figure 4.2 Process control block (PCB).

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

Implementation of Process Control Block:

Block.h

```
#include<string>
struct Block {
	std::string processState;
	int processNumber;
	std::string registers;
	std::string schedulingInfo ;
	std::string MemoryManagementInfo;
	std::string AccountingInfo;
	std::string IOstatusInfo;
	Block *programCounter;
	Block *next;
	Block(std::string pS,int pN,std::string r,std::string sI,std::string MMI,
		std::string AI,std::string IOI);
	~Block();
};
```

Block.cpp

```
#include"Block.h"
Block::Block( std::string pS,int pN,std::string r,std::string sI,
		std::string MMI,std::string AI,std::string IOI)
{
	this->processState=pS;
	this->processNumber=pN;
	this->registers=r;
	this->schedulingInfo=sI;
	this->MemoryManagementInfo=MMI;
	this->AccountingInfo=AI;
	this->IOstatusInfo=IOI;
	this->programCounter=NULL;
	this->next=NULL;
}
Block::~Block(){}
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

PCB.h

```
#include<iostream>
#include"Block.cpp"
class PCBList{
    private:
        Block *head ;
        Block *current ;

    public:
        PCBList();
        void AddProcess(std::string pS,int pN,std::string r,std::string sI,
                        std::string MMI,std::string AI,std::string IOI);
        void ProcessExecuted(int processNumber);
        void DisplayBlocks();

};
```

Pcb.cpp

```
#include"Pcb.h"
#include<iostream>

PCBList::PCBList()
{
    head=NULL;
    current=NULL;
}
void PCBList::DisplayBlocks ( )
{
    int i=1;
    Block *temp=NULL ;
    if (head == NULL)
    {
        std::cout<<"No block in the list";
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
```

```
        std::cout<<"------Block "<<i<<"------"<<
        std::endl<<"Process State: "<<temp->processState<<
        std::endl<<"Process Number: "<<temp->processNumber<<
        std::endl<<"Registers: "<<temp->registers<<
        std::endl<<"Scheduling Information: "<<temp->schedulingInfo<<
        std::endl<<"Memory Management Information: "<<temp->MemoryManagementInfo<<
        std::endl<<"Accounting Information: "<<temp->AccountingInfo<<
        std::endl<<"Input/Output status Information: "<<temp->IOstatusInfo<<std::endl<<std::endl;
        temp = temp->next;
        ++i;
     }


  }
    -
}
void PCBList::AddProcess(std::string pS,int pN,std::string r,std::string sI,
                         std::string MMI,std::string AI,std::string IOI)
{
    Block* temp = new Block(pS, pN,r,sI, MMI,AI,IOI);
    if (head == NULL)
    {
        head = temp;
    }
    else
    {
        current=head;
        while(current->next != NULL)
        {
            current =  current->next;
        }
        current->next =  temp;
    }
}
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

```
void PCBList::ProcessExecuted(int processNumber)
{
        Block *temp=NULL;
        int count =0;
        if (head == NULL)
         {
             std::cout<<"List Empty";
         }
         else
         {
             current = head;
             while(current->next!=NULL)
             {
                 if(current->processNumber==processNumber)
                 {
                     break;
                 }
                 else{
                     count++;
                     current = current->next;
                 }
             }
             current = head;
             for(int i=1;i<count;++i){
                 current=current->next;
             }

             temp = current->next;
             current->next=current->next->next;
         }
}
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

---

driver.cpp

```
#include<iostream>
#include"Pcb.cpp"
int main()
{
    PCBList processes;
    processes.AddProcess("waiting",120,"general purpose register", "priority=4",
                         "page table", "2 minutes", "1. monitor 2. printer" );
    processes.AddProcess("halted",121,"index register", "priority=6",
                         "page table", "4 minutes", "2. printer" );
    processes.DisplayBlocks();
    processes.ProcessExecuted(120);
    processes.DisplayBlocks();
}
```

Output:

```
------Block 1------
Process State: waiting
Process Number: 120
Registers: general purpose register
Scheduling Information: priority=4
Memory Management Information: page table
Accounting Information: 2 minutes
Input/Output status Information: 1. monitor 2. printer

------Block 2------
Process State: halted
Process Number: 121
Registers: index register
Scheduling Information: priority=6
Memory Management Information: page table
Accounting Information: 4 minutes
Input/Output status Information: 2. printer

------Block 1------
Process State: waiting
Process Number: 120
Registers: general purpose register
Scheduling Information: priority=4
Memory Management Information: page table
Accounting Information: 2 minutes
Input/Output status Information: 1. monitor 2. printer
```

Implementation of PCB using Queue Data Structure:

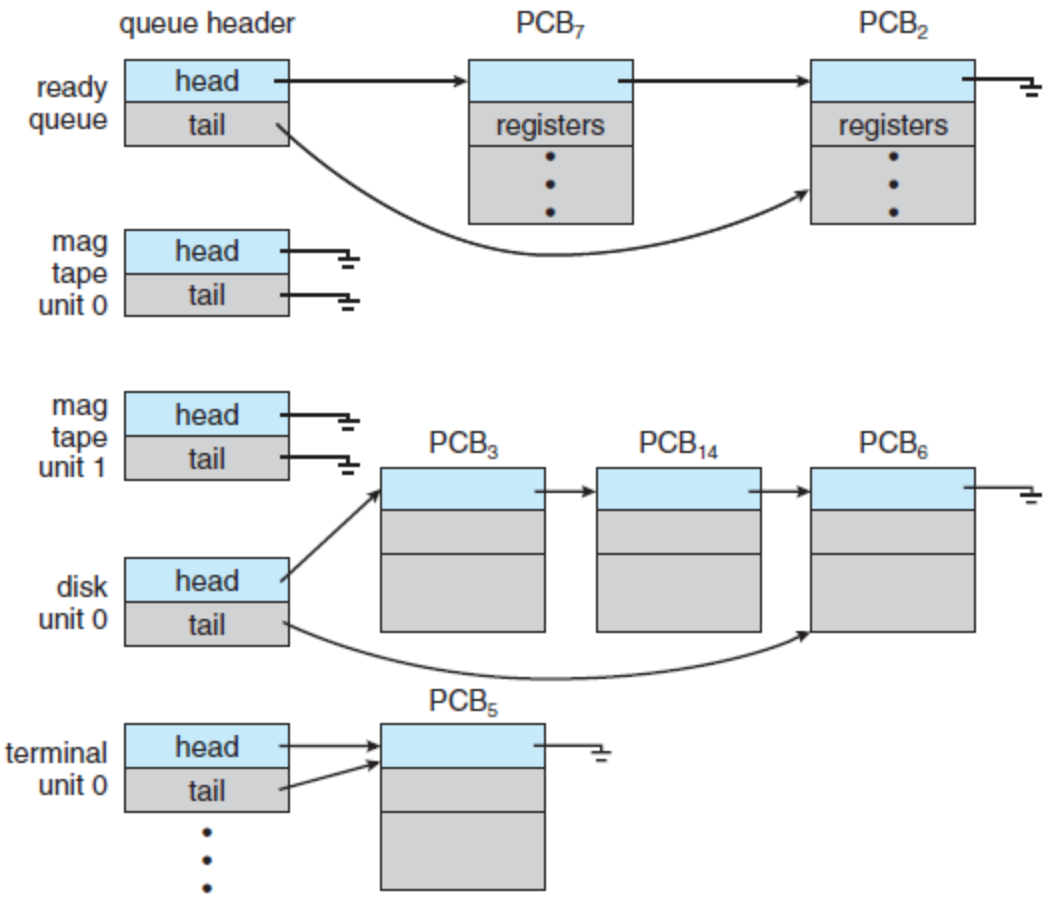

Figure 4.3 The ready queue and various I/O device queues.

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

```
#include<iostream>
struct Block {
     std::string processState;
     int processNumber;
     std::string registers;
     std::string schedulingInfo ;
     std::string MemoryManagementInfo;
     std::string AccountingInfo;
     std::string IOstatusInfo;
     Block *programCounter;
     Block *next;
     Block( std::string pS,int pN,std::string r,std::string sI,
            std::string MMI,std::string AI,std::string IOI)
    {
         this->processState=pS;
         this->processNumber=pN;
         this->registers=r;
         this->schedulingInfo=sI;
         this->MemoryManagementInfo=MMI;
         this->AccountingInfo=AI;
         this->IOstatusInfo=IOI;
         this->programCounter=NULL;
         this->next=NULL;
    }
    ~Block(){}
};

struct headNode{
    int count;
    Block *front;
    Block *rear;
    headNode():front(NULL),rear(NULL){}
};
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

```
class Queue{
private :
    headNode *head = NULL;
    int c{0};
public:

    void Enqueue(std::string pS,int pN,std::string r,std::string sI,
                 std::string MMI,std::string AI,std::string IOI);
    void Dequeue();
    void displayBlocks();

};
void Queue :: Enqueue(std::string pS,int pN,std::string r,std::string sI,
                      std::string MMI,std::string AI,std::string IOI)
{
    Block *temp = new Block(pS,pN,r,sI,MMI,AI,IOI);

    if(head==NULL)
    {
        headNode *htemp = new headNode();
        htemp->front=temp;
        htemp->rear = temp;
        htemp->count = ++c;
        head=htemp;
    }
    else{
        head->count = ++c;
        head->rear->next = temp;
        head->rear=temp;
    }
}
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

```
void Queue :: Dequeue(){

    if(head->front==NULL)
    {
      std::cout<<"Queue Underflow";
    }
    else{
        Block *temp = NULL;
        temp= head->front;
        head->count = --c;
        head->front = head->front->next;
        delete temp;
    }
}
void Queue::displayBlocks ( )
{
    int i=1;
    Block *temp=NULL ;
    if (head == NULL)
    {
        std::cout<<"No block in the list";
    }
    else
    {
        temp = head->front;
        while(temp != NULL)
        {
          std::cout<<"------Block "<<i<<"------"<<
          std::endl<<"Process State: "<<temp->processState<<
          std::endl<<"Process Number: "<<temp->processNumber<<
          std::endl<<"Registers: "<<temp->registers<<
          std::endl<<"Scheduling Information: "<<temp->schedulingInfo<<
          std::endl<<"Memory Management Information: "<<temp->MemoryManagementInfo<<
          std::endl<<"Accounting Information: "<<temp->AccountingInfo<<
          std::endl<<"Input/Output status Information: "<<temp->IOstatusInfo<<std::endl<<std::endl;
          temp = temp->next;
          ++i;
        }
    }
}
```

```
int main()
{

    Queue q;
    q.Enqueue("waiting",111,"general purpose register", "priority=4",
              "page table", "2 seconds", "monitor" );
    q.Enqueue("waiting",112,"general purpose register", "priority=2",
              "page table", "4 seconds", "printer" );
    q.Dequeue();
    q.displayBlocks();


}
```

Output:

```
------Block 1------
Process State: waiting
Process Number: 112
Registers: general purpose register
Scheduling Information: priority=2
Memory Management Information: page table
Accounting Information: 4 seconds
Input/Output status Information: printer


Process returned 0 (0x0)   execution time : 0.718 s
Press any key to continue.
```

Queue Alignment of Processes:

As processes enter the system, they are put into a job queue, which consists of all processes in the system. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue. The system also includes other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. Suppose the process makes an I/O request to a shared device, such as a disk. Since there are many processes in the system, the disk may be busy with the I/O request of some

other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a device queue.

```
#include<iostream>
#include<queue>
struct Block {
     std::string processState;
     int processNumber;
     std::string registers;
     std::string schedulingInfo ;
     std::string MemoryManagementInfo;
     std::string AccountingInfo;
     std::string IOstatusInfo;
     bool newstate=true;
     bool waiting=false;
     bool io=false;
     Block *programCounter;
     Block *next;
     Block( std::string pS,int pN,std::string r,std::string sI,
            std::string MMI,std::string AI,std::string IOI)
    {
          this->processState=pS;
          this->processNumber=pN;
          this->registers=r;
          this->schedulingInfo=sI;
          this->MemoryManagementInfo=MMI;
          this->AccountingInfo=AI;
          this->IOstatusInfo=IOI;
          this->programCounter=NULL;
          this->next=NULL;
    }
    ~Block(){}
};
class PCB{
    private:
        Block *head ;
        Block *current ;
```

```
    public:
        PCB();
        void AddProcess(std::string pS,int pN,std::string r,std::string sI,
                        std::string MMI,std::string AI,std::string IOI);
        void ProcessExecuted(int processNumber);
        void DisplayBlocks();
        Block* getheadValue();
        void SchedulingQueues(std::queue<Block*> jobQueue, std::queue<Block*> readyQueue,
                              std::queue<Block*> deviceQueue);

};

PCB::PCB()
{
    head=NULL;
    current=NULL;
}
void PCB::DisplayBlocks ( )
{
    int i=1;
    Block *temp=NULL ;
    if (head == NULL)
    {
        std::cout<<"No block in the list";
    }
    else
    {
        temp = head;
        while(temp != NULL)
      {
          std::cout<<"------Block "<<i<<"------"<<
          std::endl<<"Process State: "<<temp->processState<<
          std::endl<<"Process Number: "<<temp->processNumber<<
          std::endl<<"Registers: "<<temp->registers<<
          std::endl<<"Scheduling Information: "<<temp->schedulingInfo<<
          std::endl<<"Memory Management Information: "<<temp->MemoryManagementInfo<<
          std::endl<<"Accounting Information: "<<temp->AccountingInfo<<
          std::endl<<"Input/Output status Information: "<<temp->IOstatusInfo<<std::endl<<std::endl;
          temp = temp->next;
          ++i;
      }
```

```
    }


}
void PCB::AddProcess(std::string pS,int pN,std::string r,std::string sI,
                     std::string MMI,std::string AI,std::string IOI)
{
    Block* temp = new Block(pS, pN,r,sI, MMI,AI,IOI);


    if (head == NULL)
    {
        head = temp;
    }

    else
    {
        current=head;
        while(current->next != NULL)
        {
            current =  current->next;
        }
        current->next =  temp;

    }

}
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

```
void PCB::ProcessExecuted(int processNumber)
{

        Block *temp=NULL;
        int count =0;
        if (head == NULL)
         {
             std::cout<<"List Empty";
         }
         else
         {
             current = head;
             while(current->next!=NULL)
             {
                 if(current->processNumber==processNumber)
                 {
                     break;
                 }
                 else{
                     count++;
                     current = current->next;
                 }
             }
             current = head;
             for(int i=1;i<count;++i){
                 current=current->next;
             }

             temp = current->next;
             current->next=current->next->next;
         }
}
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

```
Block* PCB:: getheadValue(){
    return head;
}
void PCB:: SchedulingQueues(std::queue<Block*> jobQueue, std::queue<Block*> readyQueue,
                        std::queue<Block*> deviceQueue )
{
    std::string ans;
    if (jobQueue.empty()) return;
    Block *current = jobQueue.front();
    while(current!=NULL){
    if(current->newstate==true)
    {
        jobQueue.pop();
        readyQueue.push(current);
        current->newstate=false;
        current->waiting=true;
        std::cout<<"IO Devices needed?";
        std::cin>>ans;
        if(ans=="YES")
        {
            deviceQueue.push(current);
            current->io=true;
        }
         current=current->next;
    }

int main(){
    PCB processes;
    processes.AddProcess("waiting",120,"general purpose register", "priority=4",
                         "page table", "2 minutes", "1. monitor 2. printer" );
    processes.AddProcess("halted",121,"index register", "priority=6", "page table",
                         "4 minutes", "2. printer" );
    processes.DisplayBlocks();
    processes.ProcessExecuted(120);
    processes.DisplayBlocks();
    std::queue<Block*> jobQueue;
    std::queue<Block*> readyQueue;
    std::queue<Block*> deviceQueue;
    jobQueue.push(processes.getheadValue());
    processes.SchedulingQueues(jobQueue, readyQueue, deviceQueue);

}
```

Output:

```
------Block 1------
Process State: waiting
Process Number: 120
Registers: general purpose register
Scheduling Information: priority=4
Memory Management Information: page table
Accounting Information: 2 minutes
Input/Output status Information: 1. monitor 2. printer

------Block 2------
Process State: halted
Process Number: 121
Registers: index register
Scheduling Information: priority=6
Memory Management Information: page table
Accounting Information: 4 minutes
Input/Output status Information: 2. printer

------Block 1------
Process State: waiting
Process Number: 120
Registers: general purpose register
Scheduling Information: priority=4
Memory Management Information: page table
Accounting Information: 2 minutes
Input/Output status Information: 1. monitor 2. printer

IO Devices needed?yes

Process returned 0 (0x0)   execution time : 4.819 s
Press any key to continue.
```

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

Assignment # 4

Create a Scheduler class. It should enqueue the PCB objects into a Queue based on the arrival time and dequeue a PCB object whenever the CPU is free. Calculate whether CPU is free based on the allocation time + burst time of the previous PCB object. Assume CPU is free for the initial case.