```
#include <iostream>
#include <fstream>
using namespace std;

class pnode
{
public:
	int c;
	pnode *pnext;
};

class PROCESS
{
public:
	PROCESS(); //constructor
	//~PROCESS(); //destructor
	void setProcess(int data[], int);
	void AdjustTime();
	pnode *time;
	PROCESS *next;
	int process_number;
	int current_burst;
	int IOtime;
	int responsetime;
	int waitingtime;
	int turnaroundtime;
};

class QUEUE : public PROCESS
{
public:
	QUEUE();
	//	~QUEUE();
	void currentprocess(PROCESS *);
	void IOprocess(PROCESS *, int);
	void Dequeue(PROCESS *);
	void run(int&, int&, QUEUE *);
	void calculation(int&, int&);
	void display(int, QUEUE *);
	bool currentIsEmpty();
	bool waitingIsEmpty();
```

```
private:
	PROCESS *first;
};


PROCESS::PROCESS()
{
	process_number = current_burst = IOtime = 0;
	waitingtime = turnaroundtime = 0;
	responsetime = -1;
}

//PROCESS::~PROCESS()
//{
//	while (time != NULL)
//	{
//		AdjustTime();
//	}
//}

void PROCESS::setProcess(int data[], int num)
{
	process_number = num;
	pnode *ptr = new pnode;
	time = ptr;
	current_burst = data[0];
	for (int i = 0; i < 21; i++)
	{
		ptr->c = data[i];
		if (data[i + 1] == 0)
		{
			ptr->pnext = NULL;
			return;
		}
		ptr->pnext = new pnode;
		ptr = ptr->pnext;
	}

}

void PROCESS::AdjustTime() //updates burst and IO time
{
	pnode *ptr = time;
```

```
	time = time->pnext;
	delete ptr;
}

QUEUE::QUEUE()
{
	first = next = 0;
}

//QUEUE::~QUEUE()
//{
//	PROCESS *ptr = first;
//	while (ptr != NULL)
//	{
//		first = first->next;
//		ptr = NULL;
//		delete ptr;
//		ptr = first;
//	}
//}

bool QUEUE::currentIsEmpty() //checks if current its empty
{
	return first == NULL;
}

bool QUEUE::waitingIsEmpty() // checks if waiting its empty
{
	PROCESS *ptr = first;
	while (ptr != NULL)
	{
		if (ptr->IOtime > 0) return false;
		ptr = ptr->next;
	}
	return true;
}

void QUEUE::currentprocess(PROCESS *p)
{

	if (p->current_burst == 0)
	{
		p->IOtime = 0;
```

```
                p->AdjustTime();
                p->current_burst = p->time->c;
        }
        PROCESS *ptr = first;

        if (ptr == NULL)
        {
                first = p;
                first->next = 0;
                return;
        }
        PROCESS *prev = NULL;

        while (ptr != NULL)
        {
                prev = ptr;
                ptr = ptr->next;
        }
        prev->next = p;
        p->next = NULL;
}

void QUEUE::IOprocess(PROCESS *p, int timer)
{
        p->current_burst = 0;
        p->AdjustTime();
        if (p->time == NULL)
        {
                p->turnaroundtime = timer;
        }
        else
        {
                p->IOtime = p->time->c;
        }
        PROCESS *ptr = first;
        if (ptr == NULL)
        {
                first = p;
                p->next = 0;
        }
        else
        {
                p->next = ptr;
```

```
                first = p;

        }
}
void QUEUE::Dequeue(PROCESS *p)
{
        if (p == first)
        {
                first = p->next;
                return;
        }
        PROCESS *ptr = first;
        PROCESS *prev = NULL;
        while (ptr != NULL)
        {
                if (ptr == p)
                {
                        prev->next = ptr->next;
                        return;
                }
                else
                {
                        prev = ptr;
                        ptr = ptr->next;
                }
        }
}

void QUEUE::run(int &time, int &idle, QUEUE *io)
{
        PROCESS *ptr, *temp;
        int counter = 0;
        if (currentIsEmpty())
        {
                ptr = io->first;
                ptr->process_number *= -1;
                PROCESS *current = ptr;
                while (current != NULL)
                {
                        if (current->IOtime <= 0)
                        {
                                current = current->next;
                        }
```

```
                else
                {
                        break;
                }
        }
        if (current != NULL)
        {
                ptr = current->next;
                while (ptr != NULL)
                {
                        if (ptr->IOtime < current->IOtime && ptr->IOtime >0)
                        {
                                current = ptr;
                        }
                        ptr = ptr->next;
                }
                counter = current->IOtime;
                idle += counter;
                ptr = io->first;
        }
}
else
{
        counter = first->current_burst;
        ptr = io->first;
        temp = first->next;
        if (first->responsetime == -1)
                first->responsetime = time;
        while (temp != NULL)
        {
                temp->waitingtime += counter;
                temp = temp->next;
        }
        temp = first;
        Dequeue(first);
        io->IOprocess(temp, time + counter);
}
display(time, io);
time += counter;
while (ptr != NULL)
{
        ptr->IOtime -= counter;
        if (ptr->IOtime <= 0 && ptr->time != NULL)
```

```
		{
			temp = ptr->next;
			while (temp != NULL)
			{
				if (temp->time == NULL)
				{
					temp = temp->next;
				}
				else
				{
					break;
				}
			}
			ptr->waitingtime -= ptr->IOtime;
			io->Dequeue(ptr);
			currentprocess(ptr);
			ptr = temp;
		}
		else
		{
			ptr = ptr->next;
		}
	}
}

void QUEUE::display(int time, QUEUE *io) //display outpup
{

	bool idle = false;
	if ((io->first != NULL && io->first->process_number < 0) || (first != NULL &&
first->process_number < 0))
	{
		idle = true;
		if (io->first->process_number < 0)
			io->first->process_number *= -1;
		if (first != 0 && first->process_number < 0)
			first->process_number *= -1;
		cout << "Current Time: " << time << " \t\t\tProcess: [IDLE]\n----";

	}
	else
	{
```

```
                cout << "Current Time:" << time << "\t\t\t Process: P" <<
io->first->process_number;
                if (io->first->time == NULL)
                {
                        cout << " [TERMINATED]\n----------";
                }
                else
                {
                        cout << endl;
                }
        }
        for (int i = 0; i < 51; i++)
        {
                cout << "-";
        }
        cout << "\nReady Queue\tProcess\t\tCPU Burst\n";

        PROCESS *temp = first;
        if (first == NULL)
        {
                cout << "\t\tQueue Empty\n";
        }
        else
        {
                while (temp != NULL)
                {
                        if (temp->process_number != io->first->process_number || idle == 1)
                        {
                                cout << "\t\tP" << temp->process_number << "\t\t" <<
temp->current_burst << endl;
                        }
                        temp = temp->next;
                }
        }
        temp = io->first;
        cout << "\nBlocked Queue\tProcess\t\tI/O Time\n";

        if (io->first->next == NULL || io->waitingIsEmpty())
        {
                cout << "\t\tQueue Empty";

        }
        else
```

```
	{
		bool counter = true;
		while (temp != NULL)
		{
			if (counter == true)
			{
				temp = temp->next;
				while (temp != NULL)
				{
					if (temp->IOtime <= 0)
					{
						temp = temp->next;
					}
					else
					{
						break;
					}
				}
				if (temp == NULL)
				{
					cout << "\t\tQueue Empty";
					break;
				}
				else
				{
					temp = io->first;
					counter = false;
				}
			}
			if ((temp->IOtime > 0 && temp->process_number != 
io->first->process_number) || (temp->process_number == io->first->process_number && idle == 
1))
			{
				cout << "\t\tP" << temp->process_number << "\t\t" << 
temp->IOtime << endl;

			}
			temp = temp->next;
		}
	}
	cout << "\n\n\n";
}
```

```
void QUEUE::calculation(int &time, int &idle)
{
	double avgresponsetime = 0.0, avgwaitingtime = 0.0, avgTurnaroundTime = 0.0, util =
0.0;
	PROCESS *temp = first;
	cout << "\nProcess\t\tResponse Time\tWaiting Time\tTurnaround Time";

	for (int i = 0; i < 8; i++)
	{
		avgresponsetime += temp->responsetime;
		avgwaitingtime += temp->waitingtime;
		avgTurnaroundTime += temp->turnaroundtime;
		cout << "\nP" << temp->process_number << "\t\t" << temp->responsetime <<
"\t\t" << temp->waitingtime << "\t\t" << temp->turnaroundtime;
		temp = temp->next;
	}
	avgwaitingtime /= 8;
	avgresponsetime /= 8;
	avgTurnaroundTime /= 8;
	util = (1 - ((double)idle / (double)time))*100.0;
	cout << "\n\nTotal time: " << time;
	cout << "\nAverage response time: " << avgresponsetime;
	cout << "\nAverage turnaround time: " << avgTurnaroundTime;
	cout << "\nAverage waiting time: " << avgwaitingtime;

	cout << "\nCPU Utilization: " << util << "%\n\n";


}

int main(int argc, const char * argv[]) {
	PROCESS p1, p2, p3, p4, p5, p6, p7, p8;
	QUEUE readyQueue, waitingQueue;
	int timer = 0;
	int idle = 0;
	int stop = 100;
	bool end = false;

	int data1[14] = { 6, 17, 8, 19, 12, 31, 11, 18, 9, 22, 8, 26, 10, 0 };
	p1.setProcess(data1, 1);
	readyQueue.currentprocess(&p1);
```

```
int data2[18] = { 19, 38, 11, 24, 15, 21, 12, 27, 12, 34, 11, 34, 9, 29, 9, 31, 7, 0 };
p2.setProcess(data2, 2);
readyQueue.currentprocess(&p2);

int data3[18] = { 3, 37, 14, 41, 8, 30, 4, 19, 7, 33, 5, 18, 4, 26, 5, 31, 16, 0 };
p3.setProcess(data3, 3);
readyQueue.currentprocess(&p3);

int data4[16] = { 15, 35, 14, 41, 16, 45, 18, 51, 14, 61, 13, 54, 16, 61, 15, 0 };
p4.setProcess(data4, 4);
readyQueue.currentprocess(&p4);

int data5[18] = { 9, 24, 7, 21, 15, 31, 6, 26, 7, 31, 3, 18, 6, 21, 6, 33, 3, 0 };
p5.setProcess(data5, 5);
readyQueue.currentprocess(&p5);

int data6[18] = { 4, 38, 3, 41, 5, 29, 4, 26, 7, 32, 4, 22, 3, 26, 5, 22, 8, 0 };
p6.setProcess(data6, 6);
readyQueue.currentprocess(&p6);

int data7[16] = { 14, 36, 17, 31, 16, 32, 15, 41, 14, 42, 17, 39, 16, 33, 15, 0 };
p7.setProcess(data7, 7);
readyQueue.currentprocess(&p7);

int data8[18] = { 5, 14, 4, 33, 6, 31, 4, 31, 6, 27, 5, 21, 4, 19, 6, 11, 6, 0 };
p8.setProcess(data8, 8);
readyQueue.currentprocess(&p8);

while (end == false)
{
        readyQueue.run(timer, idle, &waitingQueue);

        if (readyQueue.currentIsEmpty() && waitingQueue.waitingIsEmpty())
        {
                end = true;
        }
}


cout << "\n\t\t    Processes are complete\n\n";
```

```
        waitingQueue.calculation(timer, idle);

        system("pause");
        return 0;

}
```