```
//FCFS_preemptivo
#include <pthread.h>
#include<unistd.h>
#include<bits/stdc++.h>
using namespace std;
#define N_cliente (int)1e2   //Quantidade de clientes
#define N_resposta (int)1e4  //soma máxima
#define inf 0x3f3f3f3f       //vez do escalonador
#define tempo_escalonador 0.4

volatile int buffer[N_cliente+2],idx=0;
//buffer de comunição cliente > servidor
volatile int valores_clientes[N_cliente+2],vc=0;
//Dados que seram utilizados para determinar a prioridade do clientes
volatile int retorno[N_resposta+2],check[N_resposta+2],v=0,vez;
//Resposta do servidor para o cliente
volatile int deumerda=0;
pthread_mutex_t mutex_zone=PTHREAD_MUTEX_INITIALIZER,
mutex_clients=PTHREAD_MUTEX_INITIALIZER,
mutex_pre=PTHREAD_MUTEX_INITIALIZER;
pthread_t server,cliente[N_cliente],escalonador;

void * thread_servidor(void * v){
        int resposta[N_resposta+2],count=0;
        resposta[1]=1;
        for(int i=2;i<=N_resposta;i++)
                resposta[i]=resposta[i-1]+i;
        while(true){
                if(count==N_cliente) break;
                while(idx==0);
                pthread_mutex_lock(&mutex_zone);
                count++;
                int num_cliente=buffer[--idx];
                retorno[num_cliente]=resposta[num_cliente];
                pthread_mutex_unlock(&mutex_zone);
        }
}

void * thread_cliente(void * v){
        int valor_cliente;
        clock_t ini,fim;
        pthread_mutex_lock(&mutex_clients);
        int k=rand()%N_resposta+1;
        while(check[k])
                k=rand()%N_resposta+1;
        check[k]=1;
        valores_clientes[vc++]=k;
        valor_cliente=k;
        if(vc==N_cliente) vez=inf;
```

```
	pthread_mutex_unlock(&mutex_clients);
	while(vez!=valor_cliente);
	ini=clock();
	pthread_mutex_lock(&mutex_zone);
	buffer[idx++]=valor_cliente;
	pthread_mutex_unlock(&mutex_zone);
	while(retorno[valor_cliente]==0);
	double decorrido;
	int num=retorno[valor_cliente];//num é um cópia do valor retornado
	map<int,int> fatores;
	for(int x=2;x*x<=num;x++){
		fim =clock();
		decorrido = ((double)(fim-ini))/(CLOCKS_PER_SEC);
		if(decorrido>tempo_escalonador){
			printf("Parando %d\n",valor_cliente);//Print para checar as thread paradas
			pthread_mutex_lock(&mutex_pre);
			deumerda=valor_cliente;
			pthread_mutex_unlock(&mutex_pre);
			vez=-inf;
			while(vez!=valor_cliente);
			printf("Continuando... %d\n",valor_cliente);//checa thread continuando
			ini=clock();
		}
		while(num%x==0){
			fatores[x]++;
			num/=x;
		}
	}
	if(num>1) fatores[num]++;
	printf("\nThread: %d\n",valor_cliente);
	printf("Resposta: %d == ",retorno[valor_cliente]);

	for(map<int,int>::iterator it=fatores.begin();it!=fatores.end();it++){
		if(it!=fatores.begin())
			printf(" * ");
		printf("%d^%d",it->first,it->second);
	}
	printf("\n");
	/*
	Esse printf serve para checar se a exclusão mútua funcionou
	E se todos os elementos foram printados.
	*/
	vez=inf;
}
void * thread_escalonador(void *args){
	while(vez!=inf);
	queue<int> q;
	for(int v=0;v<N_cliente;v++){
		int aux=valores_clientes[v];
```

```
                q.push(aux);
        }
        while(!q.empty()){
                while(vez!=inf){
                        pthread_mutex_lock(&mutex_pre);
                        if(deumerda){
                                q.push((int)deumerda);
                                deumerda=0;
                                pthread_mutex_unlock(&mutex_pre);
                                break;
                        }
                        pthread_mutex_unlock(&mutex_pre);
                }
                vez=q.front();
                q.pop();
        }
}

int main(){
        srand(time(NULL));
        int x;
        vez=-inf;
        pthread_create(&server,NULL,&thread_servidor,NULL);
        pthread_create(&escalonador,NULL,&thread_escalonador,NULL);
        for(x=0;x<N_cliente;x++)
                pthread_create(&cliente[x],NULL,&thread_cliente,NULL);
        pthread_join(server, NULL);
        for(x=0;x<N_cliente;x++)
                pthread_join(cliente[x], NULL);
        pthread_join(escalonador, NULL);
}
```

```
//Loteria_preemptivo
#include <pthread.h>
#include<unistd.h>
#include<bits/stdc++.h>
using namespace std;
#define N_cliente (int)1e2   //Quantidade de clientes
#define N_resposta (int)1e4  //soma máxima
#define inf 0x3f3f3f3f       //vez do escalonador
#define tempo_escalonador 0.4

volatile int buffer[N_cliente+2],idx=0;
//buffer de comunição cliente > servidor
volatile int valores_clientes[N_cliente+2],vc=0;
//Dados que seram utilizados para determinar a prioridade do clientes
volatile int retorno[N_resposta+2],check[N_resposta+2],v=0,vez;
//Resposta do servidor para o cliente
volatile int deumerda=0;
pthread_mutex_t
mutex_zone=PTHREAD_MUTEX_INITIALIZER,mutex_clients=PTHREAD_MUTEX_INITIALIZE
R,mutex_pre=PTHREAD_MUTEX_INITIALIZER;
pthread_t server,cliente[N_cliente],escalonador;
void * thread_servidor(void * v){
        int resposta[N_resposta+2],count=0;
        resposta[1]=1;
        for(int i=2;i<=N_resposta;i++)
                resposta[i]=resposta[i-1]+i;
        while(true){
                if(count==N_cliente) break;
                while(idx==0);
                pthread_mutex_lock(&mutex_zone);
                count++;
                int num_cliente=buffer[--idx];
                retorno[num_cliente]=resposta[num_cliente];
                pthread_mutex_unlock(&mutex_zone);
        }
}
void * thread_cliente(void * v){
        int valor_cliente;
        clock_t ini,fim;
        pthread_mutex_lock(&mutex_clients);
        int k=rand()%N_resposta+1;
        while(check[k])
                k=rand()%N_resposta+1;
        check[k]=1;
        valores_clientes[vc++]=k;
        valor_cliente=k;
        if(vc==N_cliente) vez=inf;
        pthread_mutex_unlock(&mutex_clients);
        while(vez!=valor_cliente);
```

```
	ini=clock();
	pthread_mutex_lock(&mutex_zone);
	buffer[idx++]=valor_cliente;pthread_mutex_unlock(&mutex_zone);
	while(retorno[valor_cliente]==0);
	double decorrido;
	int teste=retorno[valor_cliente];
	map<int,int> fatores;
	for(int x=2;x*x<=teste;x++){
		fim =clock();
		decorrido = ((double)(fim-ini))/(CLOCKS_PER_SEC);
		if(decorrido>tempo_escalonador){
			printf("Parando %d\n",valor_cliente);//Print para checar as thread paradas
			pthread_mutex_lock(&mutex_pre);
			deumerda=valor_cliente;
			pthread_mutex_unlock(&mutex_pre);
			vez=-inf;
			while(vez!=valor_cliente);
			printf("Continuando... %d\n",valor_cliente);//checa thread continuando
			ini=clock();
		}
		while(teste%x==0){
			fatores[x]++;
			teste/=x;
		}
	}
	if(teste>1) fatores[teste]++;
	printf("Thread: %d\n",valor_cliente);
	printf("Resposta: %d == ",retorno[valor_cliente]);

	for(map<int,int>::iterator it=fatores.begin();it!=fatores.end();it++){
		if(it!=fatores.begin())
			printf(" * ");
		printf("%d^%d",it->first,it->second);
	}
	printf("\n\n");
	vez=inf;
}
void * thread_escalonador(void *args){//Loteria
	while(vez!=inf);
	vector<int> Loteria;
	set<int> S;
	for(int v=0;v<N_cliente;v++){
	int aux=valores_clientes[v];
	int k=aux%4+1;
	Loteria.insert(Loteria.end(),k,aux);
	S.insert(aux);
	}
	if(Loteria.size()){
		while(S.size()>0){
```

```
                        while(vez!=inf){
                                pthread_mutex_lock(&mutex_pre);
                                if(deumerda){
                                        S.insert((int)deumerda);
                                        deumerda=0;
                                        pthread_mutex_unlock(&mutex_pre);
                                        break;
                                }
                                pthread_mutex_unlock(&mutex_pre);
                        }
                        int k=rand()%Loteria.size();
                        while(S.find(Loteria[k])==S.end())
                                k=rand()%Loteria.size();
                        S.erase(Loteria[k]);
                        vez=Loteria[k];
                }
        }
}

int main(){
        srand(time(NULL));
        int x;
        vez=-inf;
        pthread_create(&server,NULL,&thread_servidor,NULL);
        pthread_create(&escalonador,NULL,&thread_escalonador,NULL);
        for(x=0;x<N_cliente;x++)
                pthread_create(&cliente[x],NULL,&thread_cliente,NULL);
        pthread_join(server, NULL);
        for(x=0;x<N_cliente;x++)
                pthread_join(cliente[x], NULL);
        pthread_join(escalonador, NULL);
}
```

```
//Multiplas_filas
#include <pthread.h>
#include<unistd.h>
#include<bits/stdc++.h>
using namespace std;
#define N_cliente (int)1e2   //Quantidade de clientes
#define N_resposta (int)1e4  //soma máxima
#define inf 0x3f3f3f3f       //vez do escalonador

volatile int buffer[N_cliente+2],idx=0;
//buffer de comunição cliente -> servidor
volatile int valores_clientes[N_cliente+2],vc=0;
//Dados que seram utilizados para determinar a prioridade do clientes
volatile int retorno[N_resposta+2],check[N_resposta+2],v=0,vez;
//Resposta do servidor para o cliente
pthread_mutex_t
mutex_zone=PTHREAD_MUTEX_INITIALIZER,mutex_clients=PTHREAD_MUTEX_INITIALIZE
R;
pthread_t server,cliente[N_cliente],escalonador;

void * thread_servidor(void * v){
        int resposta[N_resposta+2],count=0;
        resposta[1]=1;
        for(int i=2;i<=N_resposta;i++)
                resposta[i]=resposta[i-1]+i;
        while(true){
                if(count==N_cliente) break;
                while(idx==0);
                pthread_mutex_lock(&mutex_zone);
                count++;
                int num_cliente=buffer[--idx];
                retorno[num_cliente]=resposta[num_cliente];
                pthread_mutex_unlock(&mutex_zone);
        }
}
void * thread_cliente(void * v){
        int valor_cliente;
        pthread_mutex_lock(&mutex_clients);
        int k=rand()%N_resposta+1;
        while(check[k])
                k=rand()%N_resposta+1;
        check[k]=1;
        valores_clientes[vc++]=k;
        valor_cliente=k;
        if(vc==N_cliente) vez=inf;
        pthread_mutex_unlock(&mutex_clients);
        while(vez!=valor_cliente);
        pthread_mutex_lock(&mutex_zone);
        buffer[idx++]=valor_cliente;
```

```
	pthread_mutex_unlock(&mutex_zone);
	while(retorno[valor_cliente]==0);
	printf("Thread: %d\n",valor_cliente);
	printf("Resposta: %d\n\n",retorno[valor_cliente]);
	vez=inf;
}
void * thread_escalonador(void *args){
	while(vez!=inf);
	queue<int> q[4];
	for(int s=0;s<N_cliente;s++){
		int mod=valores_clientes[s]%4;
		q[mod].push((int)valores_clientes[s]);
	}
	for(int qu=3;qu>=0;qu--){
		while(!q[qu].empty()){
			while(vez!=inf);
			vez=q[qu].front();
			//printf("%d\n",vez%4);//teste da multifila
			q[qu].pop();

		}
	}
}

int main(){
	srand(time(NULL));
	int x;
	vez=-inf;
	pthread_create(&server,NULL,&thread_servidor,NULL);
	pthread_create(&escalonador,NULL,&thread_escalonador,NULL);
	for(x=0;x<N_cliente;x++)
		pthread_create(&cliente[x],NULL,&thread_cliente,NULL);
	pthread_join(server, NULL);
	for(x=0;x<N_cliente;x++)
		pthread_join(cliente[x], NULL);
	pthread_join(escalonador, NULL);
}
```

```
//Round-Robin
#include <pthread.h>
#include<unistd.h>
#include<bits/stdc++.h>
using namespace std;
#define N_cliente (int)1e2   //Quantidade de clientes
#define N_resposta (int)1e4  //soma máxima
#define inf 0x3f3f3f3f       //vez do escalonador

volatile int buffer[N_cliente+2],idx=0;
//buffer de comunição cliente -> servidor
volatile int retorno[N_resposta+2],check[N_resposta+2],v=0,vez;
//Resposta do servidor para o cliente
pthread_mutex_t
mutex_zone=PTHREAD_MUTEX_INITIALIZER,mutex_clients=PTHREAD_MUTEX_INITIALIZE
R;
pthread_t server,cliente[N_cliente],escalonador;

struct Node{
        struct Node *prox;
        volatile int valor;
};
struct Node *valores_clientes=NULL;
volatile int vc=0;
//Dados que seram utilizados para determinar a prioridade do clientes
void push(struct Node **head_ref,volatile int data){
        struct Node *ptr1 = (struct Node *)malloc(sizeof(struct Node));
        struct Node *temp = *head_ref;
        ptr1->valor = data;
        ptr1->prox = *head_ref;
        if(*head_ref!=NULL){
                while(temp->prox != *head_ref)
                temp =temp->prox;
                temp->prox= ptr1;
        }else
                ptr1->prox=ptr1;
        *head_ref=ptr1;
}
void * thread_servidor(void * v){
        int resposta[N_resposta+2],count=0;
        resposta[1]=1;
        for(int i=2;i<=N_resposta;i++)
                resposta[i]=resposta[i-1]+i;
        while(true){
                if(count==N_cliente) break;
                while(idx==0);
                pthread_mutex_lock(&mutex_zone);
                count++;
                int num_cliente=buffer[--idx];
```

```
                retorno[num_cliente]=resposta[num_cliente];
                pthread_mutex_unlock(&mutex_zone);
        }
}
void * thread_cliente(void * v){
        int valor_cliente;
        pthread_mutex_lock(&mutex_clients);
        int k=rand()%N_resposta+1;
        while(check[k])
                k=rand()%N_resposta+1;
        check[k]=1;
        push(&valores_clientes,k);
        vc++;
        valor_cliente=k;
        if(vc==N_cliente) vez=inf;
        pthread_mutex_unlock(&mutex_clients);
        while(vez!=valor_cliente);
        pthread_mutex_lock(&mutex_zone);
        buffer[idx++]=valor_cliente;
        pthread_mutex_unlock(&mutex_zone);
        while(retorno[valor_cliente]==0);
        printf("Thread: %d\n",valor_cliente);
        printf("Resposta: %d\n\n",retorno[valor_cliente]);
        vez=inf;
}
void * thread_escalonador(void *args){
        while(vez!=inf);
        struct Node *temp=valores_clientes;
        do{
                while(vez!=inf);
                vez=temp->valor;
                temp=temp->prox;
        }while(temp!=valores_clientes);
}

int main(){
        srand(time(NULL));
        int x;
        vez=-inf;
        pthread_create(&server,NULL,&thread_servidor,NULL);
        pthread_create(&escalonador,NULL,&thread_escalonador,NULL);
        for(x=0;x<N_cliente;x++)
                pthread_create(&cliente[x],NULL,&thread_cliente,NULL);
        pthread_join(server, NULL);
        for(x=0;x<N_cliente;x++)
                pthread_join(cliente[x], NULL);
        pthread_join(escalonador, NULL);
}
```

```
//SJF
#include <pthread.h>
#include<unistd.h>
#include<bits/stdc++.h>
using namespace std;
#define N_cliente (int)1e2   //Quantidade de clientes
#define N_resposta (int)1e4  //soma máxima
#define inf 0x3f3f3f3f       //vez do escalonador

volatile int buffer[N_cliente+2],idx=0;
//buffer de comunição cliente -> servidor
volatile int valores_clientes[N_cliente+2],vc=0;
//Dados que seram utilizados para determinar a prioridade do clientes
volatile int retorno[N_resposta+2],check[N_resposta+2],v=0,vez;
//Resposta do servidor para o cliente
pthread_mutex_t
mutex_zone=PTHREAD_MUTEX_INITIALIZER,mutex_clients=PTHREAD_MUTEX_INITIALIZE
R;
pthread_t server,cliente[N_cliente],escalonador;

void merge(volatile int vetor[],int n){
        if(n==1) return ;
        int A[n/2+1],B[n/2+1];
        for(int x=0;x<n/2;x++)
                A[x]=vetor[x];
        for(int x=n/2;x<n;x++)
                B[x-(n/2)]=vetor[x];
        merge(A,n/2);
        merge(B,n-n/2);
        A[n/2]=inf;
        B[n-n/2]=inf;
        int a=0,b=0;
        for(int x=0;x<n;x++){
                if(A[a]>B[b])
                        vetor[x]=B[b++];
                else
                        vetor[x]=A[a++];
        }
}
void * thread_servidor(void * v){
        int resposta[N_resposta+2],count=0;
        resposta[1]=1;
        for(int i=2;i<=N_resposta;i++)
                resposta[i]=resposta[i-1]+i;
        while(true){
                if(count==N_cliente) break;
                while(idx==0);
                pthread_mutex_lock(&mutex_zone);
                count++;
```

```
                int num_cliente=buffer[--idx];
                retorno[num_cliente]=resposta[num_cliente];
                pthread_mutex_unlock(&mutex_zone);
        }
}
void * thread_cliente(void * v){
        int valor_cliente;
        pthread_mutex_lock(&mutex_clients);
        int k=rand()%N_resposta+1;
        while(check[k])
                k=rand()%N_resposta+1;
        check[k]=1;
        valores_clientes[vc++]=k;
        valor_cliente=k;
        if(vc==N_cliente) vez=inf;
        pthread_mutex_unlock(&mutex_clients);
        while(vez!=valor_cliente);
        pthread_mutex_lock(&mutex_zone);
        buffer[idx++]=valor_cliente;
        pthread_mutex_unlock(&mutex_zone);
        while(retorno[valor_cliente]==0);
        printf("Thread: %d\n",valor_cliente);
        printf("Resposta: %d\n\n",retorno[valor_cliente]);
        //esse printf serve para checar se a exclusão mútua funcionou
        vez=inf;
}
void * thread_escalonador(void *args){
        while(vez!=inf);
        merge(valores_clientes,N_cliente);
        for(int x=0;x<N_cliente;x++){
                while(vez!=inf);
                vez=valores_clientes[x];
        }
}
int main(){
        srand(time(NULL));
        int x;
        vez=-inf;
        pthread_create(&server,NULL,&thread_servidor,NULL);
        pthread_create(&escalonador,NULL,&thread_escalonador,NULL);
        for(x=0;x<N_cliente;x++)
                pthread_create(&cliente[x],NULL,&thread_cliente,NULL);
        pthread_join(server, NULL);
        for(x=0;x<N_cliente;x++)
                pthread_join(cliente[x], NULL);
        pthread_join(escalonador, NULL);
}
```