Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Факультет дистанционного обучения (ФДО) Кафедра
_____

ТЕМА РАБОТЫ Отчет по лабораторной работе №_1__ по дисциплине «Параллельные вычислительные процессы»

Выполнил: студент гр. _____

_____

«____» _____ 20__ г.

Проверил: к.т.н., доцент каф. АСУ ТУСУР

Романенко Владимир Васильевич

«____» _____ 20__ г.

Томск – 2022

Содержание

Лабораторная работа №1

Задание. Ресурс – оборудование (станки) на заводе. Атрибуты – наименование оборудования (станка), а также количество изделий (деталей) P (P ≥ 1), которое оно может обрабатывать одновременно. Количество станков – S (S ≥ 1). Атрибуты деталей – наименование, количество, а также список оборудования (причем заданный в требуемом порядке обработки). Алгоритмы планирования:

1. FCFS, nonpreemptive;

2. Round Robin с очередью типа FCFS, абсолютный приоритет.

Для блокировки доступа к оборудованию (станкам) использовать сеть Петри

Тема работы: «Реализация алгоритмов планирования использования процессорного времени».

Цель работы: освоить реализацию алгоритмов планирования использования ресурсов с вытесняющей и не вытесняющей многозадачностью, с абсолютным и относительным приоритетом. Освоить реализацию механизмов безопасности и синхронизации потоков, а также механизмов исключения тупиковых ситуаций.

1. Теоретическая часть

Известно достаточно большое количество самых различных алгоритмов планирования, предназначенных для достижения разных целей. Рассмотрим некоторые самые распространенные из них.

Самым простым является алгоритм планирования First-Come, First-Served (FCFS), то есть, кто первым пришел, тот первым и обслуживается. Когда процесс переходит в состояние готовности, то ссылка на его обслуживание перемещается в конец очереди. Выбор очередного процесса для выполнения реализуется из начала очереди с ликвидацией там ссылки на его обслуживание[2]. Очередь такого типа обладает в программировании специальным наименованием, а именно, First In, First Out (FIFO), то есть, первым вошел, первым вышел. Этот алгоритм выбора процесса реализует не вытесняющее планирование. Процесс, который получил в свое распоряжение процессор, может занимать его до момента истечения текущего CPU burst, то есть, промежутка времени непрерывного использования процессора. После этого для исполнения должен быть выбран очередной процесс из начала очереди. Основным достоинством алгоритма FCFS считается легкость его реализации, однако при этом у него имеется и ряд недостатков. Поскольку среднее время ожидания и среднее полное время исполнения для данного алгоритма в значительной степени зависят от порядка местоположения процессов в очереди, то процессы, которые перешли в состояние готовность, затем должны долго ожидать начала своего исполнения. По этой причине алгоритм не следует применять для систем разделения времени.

Модификацией алгоритма FCFS является алгоритм циклического планирования Round Robin (RR), то есть, это аналог вида детской карусели в США. Данный алгоритм реализуется в режиме вытесняющего планирования. Готовые к выполнению процессы проходят процесс организации в циклы и подвергаются вращению так, чтобы каждый процесс располагался около процессора в течение небольшого фиксированного отрезка времени, как

правило, это от десяти до ста миллисекунд. Пока процесс располагается рядом с процессором, он может получить процессор в свое распоряжение и выполняться[1].

Реализовать данный алгоритм можно так же, как и предыдущий, при помощи организации процессов, которые находятся в состоянии готовность, в виде очереди FIFO. Планировщик должен выбрать для очередного выполнения процесс, который расположен в начале очереди, и установить таймер для генерации прерывания по истечении определенного периода времени. При исполнении процесса возможны следующие варианты: Интервал времени CPU burst меньше или равен длительности необходимого периода времени. В таком случае процесс сам должен освободить процессор до истечения заданного периода времени, а на выполнение подается новый процесс из начала очереди и таймер начинает снова отсчет периода. Интервал времени CPU burst процесса больше выделенного периода времени. В данном случае по истечении выделенного периода времени процесс должен быть прерван таймером и помещен в конец очереди процессов, которые готовы к выполнению, а процессор предоставляется для использования процессу, расположенному в начале очереди[3].

На производительность алгоритма RR может влиять величина выделенного периода времени. При больших значениях этого периода времени, когда все процессы успевают завершить свой CPU burst до появления прерывания по времени, алгоритм RR переформируется в алгоритм FCFS. При малых величинах выделенного периода времени может возникнуть иллюзия, что любой из n процессов функционирует на собственном виртуальном процессоре, обладающем производительностью примерно 1/n от производительности фактически используемого процессора[1].

Помимо этого, при частом переключении контекста накладные расходы на выполнение переключений резко понижают производительность системы. Алгоритмы FCFS и RR имеют зависимость от порядка

местоположения в очереди процессов, которые готовы к выполнению. Когда короткие задачи располагаются в очереди ближе к ее началу, то общая производительность таких алгоритмов способна существенно возрасти.

2. Практическая часть

Листинг программы 1.

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <queue>
#include <Windows.h>
#include <iostream>
#include <mpi.h>
#include <math.h>
#include <time.h>
#define ROOT 0
#define SIZE 2

using namespace std;
int main()
{ int A[10],B[10],X[10],PR[10]={0};
  int wait [10],TurnAround[10],Complete[10];
  int i,j,smallest,count=0,time,n;
  double avg=0,tt=0;
    int tq;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    float cpu_utilisation;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
    int total_idle_time = 0;
    float throughput;
    int burst_remaining[100];
    int idx;
    int bt[20],wt[20],tat[20],avwt=0,avtat=0;
   void FindActiveTransistions();
void PrintActiveTransistions();
void Auto();
void Manually();
const int N = 500, M = 500;
long int oper;
int cond, tran;
```

```
        int Sost[N];
        int ch[M];
        int mx1[N][M];
        int mx2[M][N];
        double times[2];
        int sBuf[SIZE][SIZE/2]; int buf[M];
        int rank, size;
        MPI_Status status;
        bool fl = true;

int main(int argc, char **argv) {
   int choose;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Bcast(&oper, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
   MPI_Bcast(&choose, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
   MPI_Barrier(MPI_COMM_WORLD);
   double time = 0;

   int se[SIZE/2];
   MPI_Scatter(sBuf, 2, MPI_INT, se, 2, MPI_INT, ROOT,
MPI_COMM_WORLD);
   for (int i = 0; i < tran/SIZE; i++)
      buf[i] = 0;
   for (int i = se[0]; i < se[1]; i++)
      for (int j = 0; j < cond; j++) {
         if (mx1[j][i] >= 1)
            if (Sost[j] < mx1[j][i])
               break;
         if (j == cond - 1) {
            buf[i-se[0]] = 1;
         }
      }
   MPI_Alltoall(buf, tran/SIZE, MPI_INT, ch, tran/SIZE, MPI_INT,
MPI_COMM_WORLD);
   MPI_Barrier(MPI_COMM_WORLD);
}

void PrintActiveTransistions() {
   int cnt = 0;
   std::cout << "\n\nАктивные переходы: " << std::endl;
   for (int i = 0; i < tran; i++) {
      if (ch[i] == 1)
         std::cout << i + 1 << " ";
```

```
        else
            cnt++;
    }
    fl = true;
    if (cnt == tran) {
        std::cout << "\nАктивных переходов нет!" << std::endl;
        fl = false;
    }
}

void Manually() {

    int per;
    FindActiveTransistions();
    for (int i = 0; i < oper; i++) {
        if (rank == ROOT) {
            PrintActiveTransistions();
            if (!fl)
                break;
                else
                break;
            }
        }
        MPI_Bcast(&per, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        for (int i = 0; i < cond; i++) {
            Sost[i] -= mx1[i][per];
            Sost[i] += mx2[per][i];
        }
        FindActiveTransistions();
    }
}

void Auto() {
    int per;
    srand(time(NULL));
    for (int i = 0; i < oper; i++) {
        if (rank == ROOT) {
            if (!fl)
                break;
            while (1) {
                per = rand() % tran;
                if (ch[per] == 1)
                    break;
            }
```

```
        }
        MPI_Bcast(&per, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        for (int i = 0; i < cond; i++) {
           Sost[i] -= mx1[i][per];
           Sost[i] += mx2[per][i];
        }
        FindActiveTransistions();
    }
}

          cout<<"Enter total number of processes(maximum 20):";
          cin>>n;
           cout<<"\nEnter Process Burst Time aka DURATION \n";
          for(i=0;i<n;i++) { cout<<"P["<<i+1<<"]:";
          cin>>bt[i]; } wt[0]=0;
          for(i=1;i<n;i++) { wt[i]=0; for(j=0;j<i;j++) wt[i]+=bt[j]; }
          cout<<"\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time";
for(i=0;i<n;i++) { tat[i]=bt[i]+wt[i];
          avwt+=wt[i];
           avtat+=tat[i]; cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\
t"<<tat[i]; }
          avwt/=i;
          avtat/=i;
          PR[9]=-1;
             for(time=0;count!=n;time++)
             {
                smallest=9;
                for(i=0;i<n;i++)
                {
                   if(A[i]<=time && PR[i]>PR[smallest] && B[i]>0 )
                      smallest=i;
                }
                time+=B[smallest]-1;
                B[smallest]=-1;
                count++;
                end=time+1;
                Complete[smallest] = end;
                wait [smallest] = end - A[smallest] - X[smallest];
                TurnAround[smallest] = end - A[smallest];
             }
   for(int i = 0; i < n; i++) {
      cout<<"Enter arrival time of process "<<i+1<<": ";
      cin>>p[i].arrival_time;
      cout<<"Enter burst time of process "<<i+1<<": ";
```

```
    cin>>p[i].burst_time;
    burst_remaining[i] = p[i].burst_time;
    p[i].pid = i+1;
    cout<<endl;
}

sort(p,p+n,compare1);

queue<int> q;
int current_time = 0;
q.push(0);
int completed = 0;
int mark[100];
memset(mark,0,sizeof(mark));
mark[0] = 1;

while(completed != n) {
    idx = q.front();
    q.pop();

    if(burst_remaining[idx] == p[idx].burst_time) {
        p[idx].start_time = max(current_time,p[idx].arrival_time);
        total_idle_time += p[idx].start_time - current_time;
        current_time = p[idx].start_time;
    }

    if(burst_remaining[idx]-tq > 0) {
        burst_remaining[idx] -= tq;
        current_time += tq;
    }
    else {
        current_time += burst_remaining[idx];
        burst_remaining[idx] = 0;
        completed++;

        p[idx].completion_time = current_time;
        p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
        p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
        p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

        total_turnaround_time += p[idx].turnaround_time;
        total_waiting_time += p[idx].waiting_time;
        total_response_time += p[idx].response_time;
    }
```

```
      for(int i = 1; i < n; i++) {
         if(burst_remaining[i] > 0 && p[i].arrival_time <= current_time &&
mark[i] == 0) {
            q.push(i);
            mark[i] = 1;
         }
      }
      if(burst_remaining[idx] > 0) {
         q.push(idx);
      }

      if(q.empty()) {
         for(int i = 1; i < n; i++) {
            if(burst_remaining[i] > 0) {
               q.push(i);
               mark[i] = 1;
               break;
            }
         }
      }
   }

   avg_turnaround_time = (float) total_turnaround_time / n;
   avg_waiting_time = (float) total_waiting_time / n;
   avg_response_time = (float) total_response_time / n;
   cpu_utilisation = ((p[n-1].completion_time - total_idle_time) / (float) p[n-
1].completion_time)*100;
   throughput = float(n) / (p[n-1].completion_time - p[0].arrival_time);

   sort(p,p+n,compare2);

         cout<<"\n\nAverage Waiting Time:"<<avwt;
         cout<<"\nAverage Turnaround Time:"<<avtat;
          return 0; }


cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\
t"<<p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\
t"<<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
   }
   cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
   cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
   cout<<"Average Response Time = "<<avg_response_time<<endl;
   cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
   cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;
```

}

```
#P      AT      BT      ST      CT      TAT     WT      RT

1       1       1       1       2       1       0       0

2       1       1       2       3       2       1       1

Average Turnaround Time = 1.50
Average Waiting Time = 0.50
Average Response Time = 0.50
CPU Utilization = 66.67%
Throughput = 1.00 process/unit time

--------------------------------
Process exited after 8.873 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1-Реализация программы

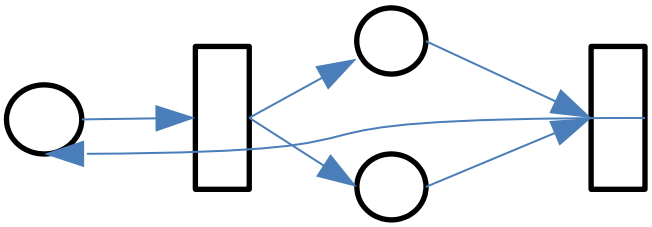

Рисунок 2-Сеть Петри

Заключение

В ходе лабораторной работы была освоена реализация алгоритмов планирования использования ресурсов с вытесняющей и не вытесняющей многозадачностью, с абсолютным и относительным приоритетом. Также была изучена реализация механизмов безопасности и синхронизации потоков, а также механизмов исключения тупиковых ситуаций. Все результаты работы представлены в скриншотах, которые показывают работу разработанного алгоритма.

Список использованных источников

1. Таненбаум, Э. Современные операционные системы / Э. Таненбаум, Х. Бос. – 4-е изд. – СПб. : Питер, 2018. – 1120 с.

2. Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core / Э. Троелсен, Ф. Джепикс. – 8-е изд. – СПб. : Питер, 2018. – 1328 с.

3. Харт, Дж. Системное программирование в среде Win32 / Дж. Харт. – 3-е изд. – М. : Вильямс, 2005. – 586 с.