Лабораторные работы по курсу
Операционные системы

Лабораторная работа 5
«Планирование процессов»

Москва, 2023

Оглавление

1. Понятие процесса 3
2. Планирование процессов 4
3. Алгоритмы планирования процессов 4
3.1. Алгоритм FCFS 4
3.2. Алгоритм SJF 5
3.3. Алгоритм RR 6
4. Практическая часть 7
5. Приложение 8

1. Понятие процесса

На прошлой лабораторной работе было введено понятие процесса. Напомним, что **процесс** – *программа во время исполнения или объект, которому выделяются ресурсы вычислительной системы, такие как процессорное время, память и т.д*.

При некоторых допущениях можно считать, что все, что выполняется в вычислительных системах (не только программы пользователей, но и, возможно, определенные части операционных систем), организовано как набор процессов. Понятно, что реально на однопроцессорной компьютерной системе в каждый момент времени может исполняться только один процесс. Для мультипрограммных вычислительных систем псевдопараллельная обработка нескольких процессов достигается с помощью переключения процессора с одного процесса на другой. Пока один процесс выполняется, остальные ждут своей очереди на получение процессора.

Процесс не может сам перейти из одного состояния в другое. Изменением состояния процессов занимается операционная система, совершая операции над ними. Количество таких операций в нашей модели пока совпадает с количеством стрелок на диаграмме состояний, изображенной на Рисунок 1.

Рисунок 1 Граф состояний процесса

Операции создания и завершения процесса являются одноразовыми, так как применяются к процессу не более одного раза (некоторые системные процессы никогда не завершаются при работе вычислительной системы). Все остальные операции, связанные с изменением состояния процессов, будь то запуск или блокировка, как правило, являются многоразовыми.

2. Планирование процессов

Планирование использования процессора впервые возникает в мультипрограммных вычислительных системах, где в состоянии готовность могут одновременно находиться несколько процессов. Именно для процедуры выбора из них одного процесса, который получит процессор в свое распоряжение, т.е. будет переведен в состояние исполнение, используется словосочетание планирование процессов.

Планирование использования процессора выступает в качестве краткосрочного планирования процессов. Оно проводится, к примеру, при обращении исполняющегося процесса к устройствам ввода-вывода или просто по завершении определенного интервала времени. Поэтому краткосрочное планирование осуществляется весьма часто, как правило, не реже одного раза в 100 миллисекунд. Выбор нового процесса для исполнения оказывает влияние на функционирование системы до наступления очередного аналогичного события, т. е. в течение короткого промежутка времени, что и обусловило название этого уровня планирования - краткосрочное.

В некоторых вычислительных системах бывает выгодно для повышения их производительности временно удалить какой-либо частично выполнившийся процесс из оперативной памяти на диск, а позже вернуть его обратно для дальнейшего выполнения. Такая процедура в англоязычной литературе получила название swapping, что можно перевести на русский язык как перекачка, хотя в профессиональной литературе оно употребляется без перевода - свопинг.

3. Алгоритмы планирования процессов

3.1.Алгоритм FCFS

FCFS (first come – first serve; первым пришел – первым обслуживается, основан на принципе FIFO) – простейший алгоритм, работа, которой понятна из ее названия. Это алгоритм без вытеснения, то есть процесс, выбранный для выполнения на ЦП, не прерывается, пока не завершится (или не перейдет в состояние ожидания по собственной инициативе). FCFS обеспечивает минимум накладных расходов. Поэтому алгоритм FCFS считается лучшим для длинных процессов. Существенным достоинством этого алгоритма наряду с его простотой является то обстоятельство, что FCFS гарантирует отсутствие бесконечного откладывания процессов: любой поступивший в систему процесс будет, в конце концов, выполнен независимо от степени загрузки системы. (Рисунок 2)

2. Планирование по принципу FIFO (First Input First Output)

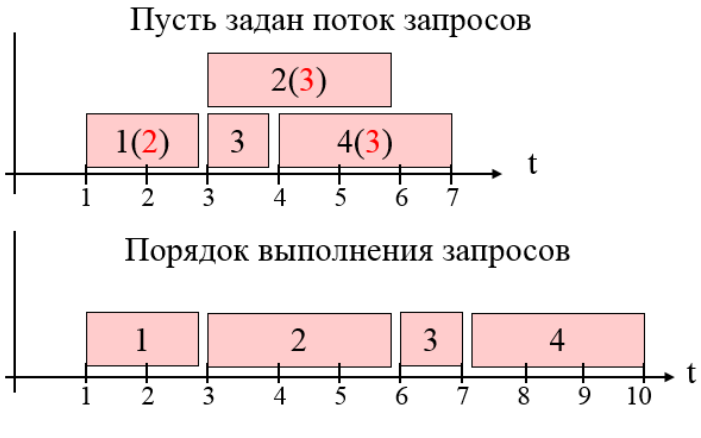
Из очереди выбирается тот процесс, который раньше пришел в систему.
БЕЗ ПЕРЕКЛЮЧЕНИЯ.

ОСОБЕННОСТИ:

Простота реализации.

Длинные процессы блокируют ЦП.

Нельзя использовать в интерактивных системах



$t_{real} = t_{out} - t_{inp}$ $\Delta = t_{real} - t_{exe}$

Pr	t_{inp}	t_{exe}	t_{out}	t_{real}	Δ
1	1	2	3	2	0
2	3	3	6	3	0
3	3	1	7	4	3
4	4	3	10	6	3

Рисунок 2 Планирование по принципу FCFS

3.2.Алгоритм SJF

SJF (shortest job first – самая короткая работа – первой) – невытесняющий алгоритм, в котором наивысший приоритет имеет самый короткий процесс. Для того чтобы применять этот алгоритм, должна быть известна длительность процесса – задаваться пользователем или вычисляться методом экстраполяции. Для коротких процессов SJN обеспечивает лучшие показатели, чем RR, как по потерянному времени, так и по штрафному отношению. SJN обеспечивает максимальную пропускную способность системы – выполнение максимального числа процессов в единицу времени, но показатели для длинных процессов значительно худшие, а при высокой степени загрузки системы активизация длинных процессов может откладываться до бесконечности. (Рисунок 3)

Рисунок 3 Планирование по принципу SJF

3.3.Алгоритм RR

RR (round robin – карусель) – простейший алгоритм с вытеснением. Процесс получает в свое распоряжение ЦП на некоторый квант времени Q (в простейшем случае размер кванта фиксирован). Если за время Q процесс не завершился, он вытесняется из ЦП и направляется в конец очереди готовых процессов, где ждет выделения ему следующего кванта, и т.д. Показатели эффективности RR существенно зависят от выбора величины кванта Q. RR обеспечивает наилучшие показатели, если длительность большинства процессов приближается к размеру кванта, но не превосходит его. Тогда большинство процессов укладываются в один квант и не становятся в очередь повторно. При величине кванта, стремящейся к бесконечности, RR вырождается в FCFS. При Q, стремящемся к 0, накладные расходы на переключение процессов возрастают настолько, что поглощают весь ресурс ЦП. (Рисунок 4)

Рисунок 4 Планирование по принципу RR

4. Практическая часть

В приложении к лабораторной работе находятся исходные коды программ, моделирующие алгоритмы планирования.

В соответствии с заданным вариантом необходимо выполнить следующие задания:

1. Скомпилируйте исходные коды каждого из алгоритмов.
2. Запустите программы и задайте следующие параметры:
   Число процессов – 4
   Время выполнения процесса 0 – 3
   Время выполнения процесса 1 – 4
   Время выполнения процесса 2 – 25
   Время выполнения процесса 3 – 3

   (Для RR размер кванта = 1)

   Проанализируйте результаты и сделайте выводы. Какой из алгоритмов в данном случае более эффективный? Скрины работы программы добавьте в отчет.

3. Запустите программы и задайте следующие параметры:
   Число процессов – 4

Время выполнения процесса 0 – 3
Время выполнения процесса 1 – 4
Время выполнения процесса 2 – 6
Время выполнения процесса 3 – 7

(Для RR размер кванта = 1)

Проанализируйте результаты и сделайте выводы. Какой из алгоритмов в данном случае более эффективный? Скрины работы программы добавьте в отчет.

4. В соответствии с вашим вариантом, доработайте одну из программ так, чтобы время прихода задачи в систему можно было делать произвольным. Проверьте работоспособность программы на примере с лекции (см. слайды презентации выше)

   Варианты:

Номер бригады	Алгоритм планирования
1	RR
2	RR
3	SJF
4	RR
5	FCFS
6	SJF
7	FCFS
8	SJF

5. Приложение

FCFS.c

```
#include <stdio.h>

int main()
{
    int bt[20], wt[20], tat[20], i, n;
    float wtavg, tatavg;
    printf("\nEnter the number of processes -- ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
            printf("\nEnter Burst Time for Process %d -- ", i);
            scanf("%d", &bt[i]);

    }
    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];
    for(i=1;i<n;i++)
    {
            wt[i] = wt[i-1] + bt[i-1];
            tat[i] = tat[i-1] + bt[i];
            wtavg = wtavg + wt[i];
            tatavg = tatavg + tat[i];

    } printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for(i=0;i<n;i++)
            printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time -- %f", wtavg/n);
```

```
    printf("\nAverage Turnaround Time -- %f\n", tatavg/n);
    return 0;
}
```

SJF.c

```
#include <stdio.h>

int main()
{
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float wtavg, tatavg;
    printf("\nEnter the number of processes -- ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
    {
            p[i]=i;
            printf("Enter Burst Time for Process %d -- ", i);
            scanf("%d", &bt[i]);

    }
    for(i=0;i<n;i++)
    for(k=i+1;k<n;k++)
    if(bt[i]>bt[k])
    {
            temp=bt[i];
            bt[i]=bt[k];
            bt[k]=temp;
            temp=p[i];
            p[i]=p[k];
            p[k]=temp;

    }
    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];
    for(i=1;i<n;i++)
    {
            wt[i] = wt[i-1] +bt[i-1];
            tat[i] = tat[i-1] +bt[i];
            wtavg = wtavg + wt[i];
            tatavg = tatavg + tat[i];

    } printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for(i=0;i<n;i++)
    printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time -- %f", wtavg/n);
    printf("\nAverage Turnaround Time -- %f", tatavg/n);
     return 0;
}
```

RR.c

```
#include <stdio.h>

int main()
{
    int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
    float awt=0,att=0,temp=0;
    printf("Enter the no of processes -- ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            printf("\nEnter Burst Time for process %d -- ", i+1);
            scanf("%d",&bu[i]); ct[i]=bu[i];

    }
    printf("\nEnter the size of time slice -- ");
    scanf("%d",&t);
    max=bu[0];
```

```
    for(i=1;i<n;i++)
    if(max<bu[i])
    max=bu[i];
    for(j=0;j<(max/t)+1;j++)
    for(i=0;i<n;i++)
    if(bu[i]!=0)
    {
            if(bu[i]<=t)
            {
                    tat[i]=temp+bu[i];
                    temp=temp+bu[i];
                    bu[i]=0;
            }
            else
            {
                    bu[i]=bu[i]-t;
                    temp=temp+t;
            }
    }
    for(i=0;i<n;i++)
    {
           wa[i]=tat[i]-ct[i];
           att+=tat[i];
           awt+=wa[i];
    }
    printf("\nThe Average Turnaround time is -- %f",att/n);
    printf("\nThe Average Waiting time is -- %f ",awt/n);
    printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
    for(i=0;i<n;i++)
    printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
return 0;
}
```