Trabalho de SO

Algoritmos de escalonamento

Por Carlos Gabriel Silva Stedile

Resumo:

Esse trabalho é um estudo dentro da área de sistemas operacionais de algoritmos de escalonamento de processos. Os algoritmos estudados nesse trabalho são: FCFS(First Come First Served), SJF(Shortest Job First), por prioridade, Round Robin, Múltiplas filas e Múltiplas filas com realimentação. Alguns dos algoritmos citados anteriormente não estão sendo utilizados mais pelos sistemas operacionais modernos como algoritmos de escalonamento em sistemas operacionais de propósito geral, como o Round Robin e FCFS. Para o entendimento dos algoritmos, foram geradas tabelas com dados de processos imaginários e gráficos simulando o funcionamento dos algoritmos perante os dados fictícios. No decorrer da pesquisa, foi perceptível a diferença de desempenho entre os algoritmos e foram constatados cenários em que alguns algoritmos eram perfeitamente explicáveis, como utilizar o FCFS em sistemas de tempo real onde a ordem de chegada dos processos é de crucial importância, enquanto outros apresentavam a pior solução possível, onde utilizar o Round Robin nos mesmos sistemas de tempo real pode acabar causando uma catástrofe com relação ao próprio sistema, provando assim a utilidade de conhecer e entender os diversos algoritmos de escalonamento de processos estudados. Os algoritmos implementados foram feitos utilizando a linguagem de programação C, desenhada originalmente para o desenvolvimento de sistemas operacionais.

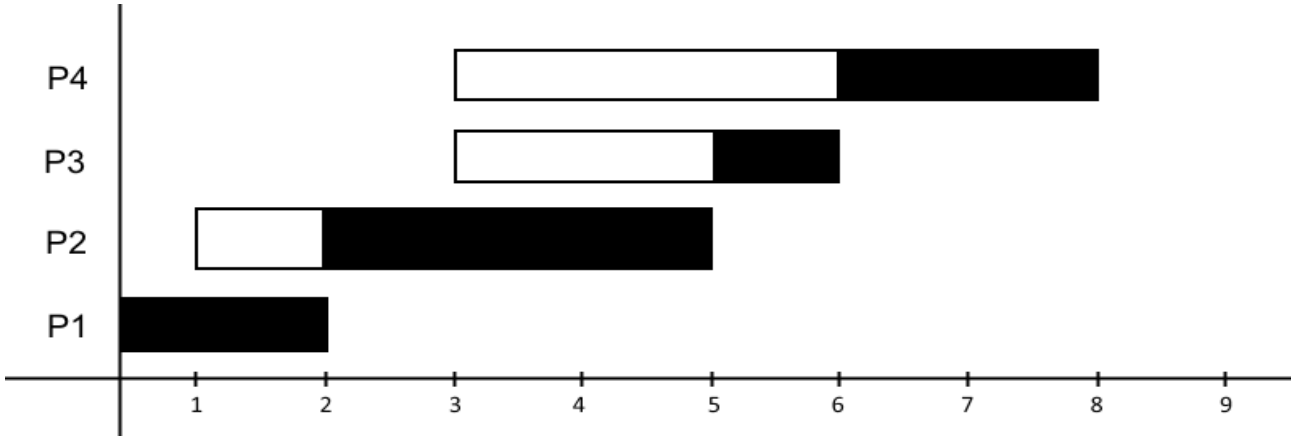
FCFS – First Come First Served

O FCFS é um algoritmo de fila simples, fazendo com que o primeiro processo na fila seja o primeiro a executar, sem ordem de prioridade o qual é de extrema importancia para sistemas operacionais de tempo real. Esse algoritmo executa os processo como um todo(ou seja, ele não é preemptivo) e quando o processo for terminado, é executado o próximo processo pronto na fila de prontos. Como todos os processos são executados, independente do seu tamanho, o algoritmo FCFS não tem *starvation*(processo nunca ser executado), exceto se um processo execute um loop infinito. O algoritmo FCFS não pode garantir um tempo de resposta rápido pois é extremamente sensível a ordem de chegada de cada processo. Caso aconteça de processos com maior tempo de execução chegarem primeiro, acarreta o aumento do tempo médio de espera dos processos na fila de espera e seu *turnaround*(tempo transcorrido desde o momento em que o software entra e o instante em que termina sua execução).

Para dar um exemplo do funcionamento do algoritmo FCFS, consideremos os processos na fila de processos prontos, com suas durações previstas de processamento e datas de ingresso no sistema, descritas na tabela a seguir:

Processos	P1	P2	P3	P4
Ingresso	0	1	3	3
Duração	2	3	1	2

O gráfico da execução desses processos:



Onde a parte branca representa o tempo de espera desde que o processo entrou na fila de prontos e a parte preta o tempo em execução pela CPU.
É possível fazer o calculo do tempo médio de execução com a função:

$$T_t = \frac{(T_e(P1) + T_e(P2) + T_e(P3) + ... + T_e(Pn))}{n}$$

Onde T_t é o tempo médio de execução, T_e é o tempo do processo ter finalizado menos o tempo o qual ingressou na fila de prontos.

E fazer o calculo do tempo médio de espera:

$$T_w = \frac{(T_s(P1) + T_s(P2) + T_s(P3) + ... + T_s(Pn))}{n}$$

Onde T_w é o tempo médio de espera, T_s é o tempo do processo ter iniciado o processamento menos o tempo o qual ingressou na fila de prontos.

Levando em conta os processos já citados, o tempo médio de execução é:

$$T_t = \frac{(T_e(P1)+T_e(P2)+T_e(P3)+T_e(P4))}{4} = \frac{((2-0)+(5-1)+(6-3)+(8-3))}{4} = \frac{(2+4+3+5)}{4} = \frac{14}{4} = 3.5s$$

E o tempo médio de espera:

$$T_w = \frac{(T_s(P1)+T_s(P2)+T_s(P3)+T_s(P4))}{4} = \frac{((0-0)+(2-1)+(5-3)+(6-3))}{4} = \frac{(0+1+2+3)}{4} = \frac{6}{4} = 1.5s$$

Um exemplo do algoritmo pode ser encontrado no anexo 1, após as referências.

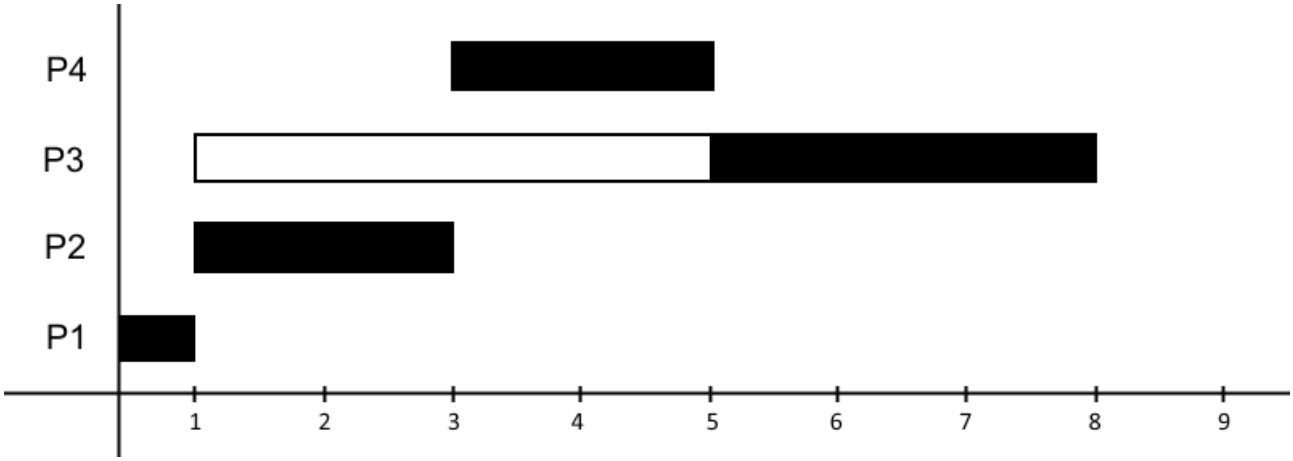
SJF – Shortest Job First

O SJF é um algoritmo de escalonamento de processos baseado no tempo que o mesmo levaria para ser executado, sempre executando primeiro o processo mais rápido na fila de prontos.
Para o funcionamento desse algoritmo, é necessário saber previamente quanto tempo o processo levara para ser executado. O SJF executa os processos como um todo(ou seja, ele não é preemptivo) e quando o processo em execução terminar, segue para o próximo processo pronto com menor tempo de execução. Porem, como sempre o processo mais rápido é executado, pode acontecer o fenômeno de *starvation*(processo nunca ser executado) em processos que levariam um longo tempo no processador.

Um exemplo do funcionamento do algoritmo SJF, considerando os processos na fila de processos prontos, com suas durações previstas de processamento e datas de ingresso no sistema, descritas na tabela a seguir:

Processos	P1	P2	P3	P4
Ingresso	0	1	1	3
Duração	1	2	3	2

O gráfico da execução desses processos:



Onde a parte branca representa o tempo de espera desde que o processo entrou na fila de prontos e a parte preta o tempo em execução pela CPU.
É possivel fazer o calculo do tempo médio de execução com a mesma função que o FCFS utiliza:

$$T_t = \frac{(T_e(P1) + T_e(P2) + T_e(P3) + ... + T_e(Pn))}{n}$$

E fazer o calculo do tempo médio de espera:

$$T_w = \frac{(T_s(P1) + T_s(P2) + T_s(P3) + ... + T_s(Pn))}{n}$$

Utilizando as formulas dadas, o tempo médio de execução e espera são, consecutivamente:

$$T_t = \frac{(T_e(P1) + T_e(P2) + T_e(P3) + T_e(P4))}{4} = \frac{((1-0) + (3-1) + (8-1) + (5-3))}{4} = \frac{(1+2+7+2)}{4} = \frac{12}{4} = 3s$$

$$T_w = \frac{(T_s(P1) + T_s(P2) + T_s(P3) + T_s(P4))}{4} = \frac{((0-0) + (1-1) + (5-1) + (3-3))}{4} = \frac{(0+0+4+0)}{4} = \frac{4}{4} = 1s$$

Comparando a execução do algoritmo SJF com o algoritmo FCFS, é possível ver uma redução no tempo de execução e espera médios.

Um exemplo do algoritmo pode ser encontrado no anexo 2, após as referências.

Prioridade

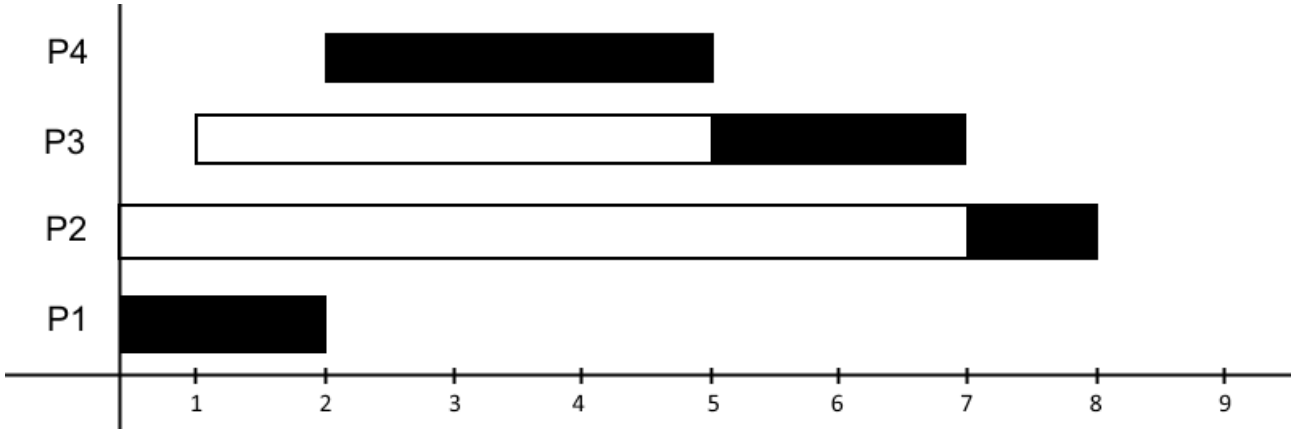
O escalonamento por prioridade gira em torno de escolher uma maneira de ordenar a fila de processo prontos para escolher qual será o próximo processo a ser executado. Escalonar por prioridades define um modelo genérico de escalonamento o qual permite modelar várias abordagens.

Para o seu funcionamento, cada processo precisa ser associado com uma prioridade, o que geralmente é feito atraves de números inteiros. Com os valores de prioridade definidos, os mesmos são utilizados para decidir qual será o próximo processo a ser executado. Não podemos esquecer que o escalonamento por prioridade pode ou não ser preemptivo, tudo depende de como é implementado.

Para facilitar seu entendimento, um exemplo do funcionamento do algoritmo de escalonamento por prioridades de maneira não preemptiva, as quais foram definidas por numeros inteiros descritos na tabela a seguir:

Processos	P1	P2	P3	P4
Ingresso	0	0	1	2
Duração	2	1	2	3
Prioridade	4	1	2	3

Que gera o seguinte grafico:



Onde a parte branca representa o tempo de espera desde que o processo entrou na fila de prontos e a parte preta o tempo em execução pela CPU.
Para fazer o calculo do tempo médio de espera e de execução, é possivel através das seguintes funções:

$$T_t = \frac{(T_e(P1) + T_e(P2) + T_e(P3) + ... + T_e(Pn))}{n}$$

$$T_w = \frac{(T_s(P1) + T_s(P2) + T_s(P3) + ... + T_s(Pn))}{n}$$

Aplicando as funções nos dados dos processos obtemos respectivamente o seu tempo médio de execução e espera:

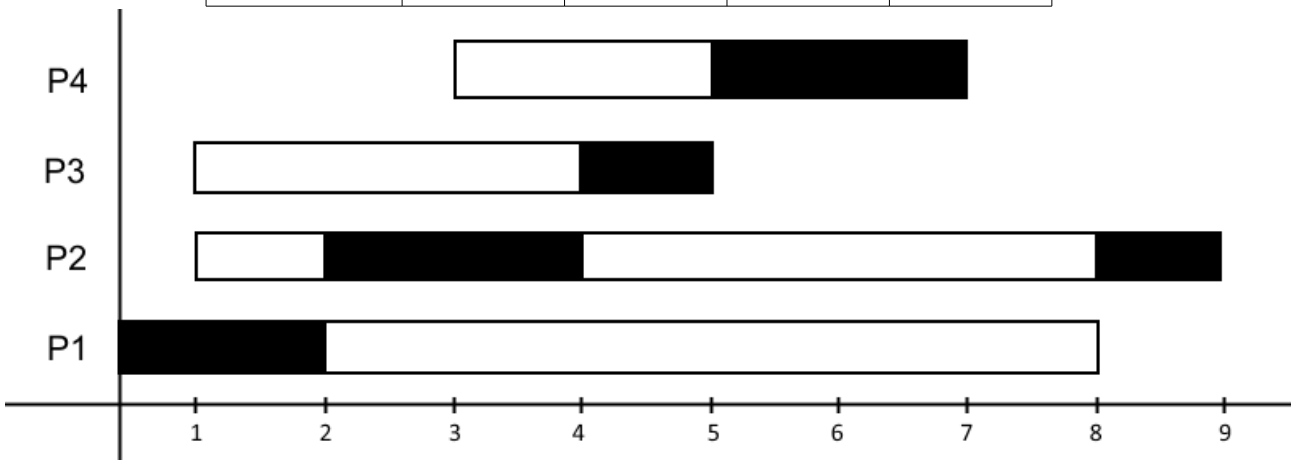
$$T_t = \frac{(T_e(P1) + T_e(P2) + T_e(P3) + T_e(P4))}{4} = \frac{((2-0) + (8-0) + (7-1) + (5-2))}{4} = \frac{(2+8+6+3)}{4} = \frac{19}{4} = 4.75\,s$$

$$T_w = \frac{(T_s(P1) + T_s(P2) + T_s(P3) + T_s(P4))}{4} = \frac{((0-0) + (7-0) + (5-1) + (3-3))}{4} = \frac{(0+7+4+0)}{4} = \frac{11}{4} = 2.75\,s$$

Round-Robin

Round-Robin é um algoritmo de escalonamento de processos preemptivo(que é capaz de parar um processo em execução para executar outro o qual esteja pronto) que atribui frações de tempo para cada processo em partes iguais e de forma circular, manipulando todos os processos sem prioridade;
Para esse algoritmo funcionar, é necessario estabelecer uma unidade de tempo chamada quantum, que é definida pelo sistema operacional, a qual determina o tempo entre cada sinal de interrupção. Caso o processo sendo executado não tenha sido terminado após passar um quantum, ele sera interrompido e voltara a ser executado de onde parou no proximo agendamento.
O seu funcionamento pode ser exemplificado apartir da tabela e grafico a seguir:

Quantum = 2				
Processos	P1	P2	P3	P4
Ingresso	0	1	1	3
Duração	3	3	1	2



Um exemplo do algoritmo pode ser encontrado no anexo 3, após as referências.

Multiplas Filas

Esse é um algortimo de escalonamento que, como o próprio nome já diz, utiliza várias filas de processos, com cada fila tendo uma prioridade diferente. Os processos são designados a uma fila de acordo com sua demanda de processamento, sendo que os de menor processamento tem maior prioridade. Os primeiros processos a serem executados de acordo com essa ordem são os processos que utilizam menos processamento, como entrada de mouse ou teclado.

Multiplas Filas com realimentação

Parecido com o algoritmo de multiplas filas, só que nesse algoritmo, o processo não permanece na mesma fila até o fim de sua execução, pois o SO faz um ajuste dinâmico para ajustar os processos em relação ao comportamento do sistema

Conclusões:

Com esse trabalho, é possível perceber que cada algoritmo de escalonamento tem características próprias, definindo assim, sua utilidade. Como utilizar o FCFS em sistemas de tempo real, aonde a execução do algoritmo por sua ordem de chegada é de crucial importância, e o Round Robin quando se tem um único núcleo e processos muito grandes precisam ser executados de forma semi paralela e em lugares que não importam tanto a prioridade. Foi notado também que alguns algoritmos de escalonamento possuem condições que podem levar processos a passar pelo efeito de *starvation*.

Referencias:

https://pt.wikipedia.org/wiki/FIFO_(escalonamento)
http://www.univasf.edu.br/~andreza.leite/aulas/SO/ProcessosEscalonamento.pdf
https://www.thecrazyprogrammer.com/2014/11/c-cpp-program-for-first-come-first-served-fcfs.html
https://www.thecrazyprogrammer.com/2014/08/c-program-for-shortest-job-first-sjf.html
https://www.thecrazyprogrammer.com/2015/09/round-robin-scheduling-program-in-c.html
https://pt.wikipedia.org/wiki/Shortest_job_first
https://joaoricardao.files.wordpress.com/2012/07/algoritmos_escalonamento.pdf
http://www.gileduardo.com.br/ifpr/so/downloads/so_aula07.pdf
https://pt.wikipedia.org/wiki/Round-robin
https://pt.wikipedia.org/wiki/M%C3%BAltiplas_filas

ANEXOS:

ANEXO 1:

```
//Algoritmo de FCFS – adaptado de:
//https://www.thecrazyprogrammer.com/2014/11/c-cpp-program-for-first-come-first-served-fcfs.html
#include<stdio.h>
int main(){
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Numero total de processos(maximo 20):");
    scanf("%d",&n);
    printf("\nEntre com o burst time(tempo de execucao) de cada processo\n");
    for(i=0;i<n;i++){
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }
    wt[0]=0;
    for(i=1;i<n;i++){
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }
    printf("\nProcesso\tBurst time\tTempo de espera\t\tTempo de Turnaround");
    for(i=0;i<n;i++){
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("\n\nTempo medio de espera: %d",avwt);
    printf("\nTempo medio de Turnaround: %d",avtat);
    return 0;
}
```

ANEXO 2:

```
//Algoritmo de SJF – adaptado de:
//https://www.thecrazyprogrammer.com/2014/08/c-program-for-shortest-job-first-sjf.html
#include<stdio.h>
void main(){
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Numero total de processos(maximo 20):");
   scanf("%d",&n);
   printf("\nEntre com o burst time(tempo de execucao) de cada processo\n");
   for(i=0;i<n;i++){
      printf("p%d:",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }
   for(i=0;i<n;i++){
      pos=i;
      for(j=i+1;j<n;j++){
         if(bt[j]<bt[pos])
            pos=j;
      }
      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;
      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }
   wt[0]=0;
   for(i=1;i<n;i++){
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
      total+=wt[i];
   }
   avg_wt=(float)total/n;
   total=0;
   printf("\nProcesso\tBurst time\tTempo de espera\t\tTempo de Turnaround");
   for(i=0;i<n;i++){
      tat[i]=bt[i]+wt[i];
      total+=tat[i];
      printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }
   avg_tat=(float)total/n;
   printf("\n\nTempo medio de espera: %f",avg_wt);
   printf("\nTempo medio de Turnaround: %f\n",avg_tat);
}
```

ANEXO 3:

```
//Codigo retirado de:
//https://www.thecrazyprogrammer.com/2015/09/round-robin-scheduling-program-in-c.html
#include<stdio.h>
int main(){
  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Numero total de processos:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++){
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Entre com o tempo de Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcesso\t|Turnaround     |Tempo de espera\n\n");
  for(time=0,count=0;remain!=0;){
    if(rt[count]<=time_quantum && rt[count]>0){
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0){
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1){
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
      wait_time+=time-at[count]-bt[count];
      turnaround_time+=time-at[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(at[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
  return 0;
}
```