Experiments

Marc Cabezas

24 de Novembre de 2016

1 Workload 1

1.1 Descripcio del proces

El primer workload prova el rendiment de les diferents polítiques de planífi cacio amb processos intensius en calcul, amb molt poca entrada/sortida. El proces pare crea un fillll i, que tambe crea un fill (3 procesos en total) i tots executen un gran nombre (igual per a tots) d'iteracions d'un bucle que no fa res (funcio prueba_estres()).

Round Robin

Aquesta politica de planíficacio reparteix equitativament els recursos entre els processos, i donat que tots executen el mateix codi tots ocuparan la CPU un nombre molt semblant de ticks, i esperaran a la cua de ready un temps equivalent. El proces idle no entra en cap moment en execucio donat que mai estan tots els procesos bloquejats.

First Come First Served

Amb aquesta politica de planíficacio els procesos fills nomes abandonen la CPU quan acaben la seva execucio, i tot i que aixo no afecta al temps total si que fa variar el temps que un proces individual pasa a la cua de ready (el primer fill no esta parat en cap moment, pero el segon ha d'esperar que aquest acabi i el pare ha d'esperar que acabin els 2).

1.2 Codi

```
void prueba_estres(int n) {

    int a = 10;

    for(int i = 0; i < n; ++i) {

        a = a*2;

    }
}
```

```
void workload1() {

    int pid1 = fork();
    if(pid1 = 0) {
        int pid2 = fork();
        if(pid2 = 0) {
            prueba_estres(2000000);
            print_status();
            exit();
        }
        else {
            prueba_estres(2000000);
            print_status();
            exit();
        }
    }
    else {
        prueba_estres(2000000);
        print_status();
    }
}
```

2 Workload 2

2.1 Descripcio del proces

Aquest experiment prova un proces intensiu en Entrada/Sortida (de fet nomes en entrada, ja que la sortida no bloqueja realment la CPU). Cada proces executa una lectura que el bloqueja durant 1500 tics, realitza una petita rafaga de CPU i despres torna a llegir.

Round Robin

: En aquest cas la politica de plani cacio casi ni actua, ja que cap proces executa calculs llargs i per tant mai s'expulsa un proces de la CPU per que se li ha acabat el quantum.

First Come First Served

: El resultat es similar a l'obtingut en RR, ja que tots els procesos estan fortament lligats a l'entrada sortida, per tant els procesos hauran d'esperar a que els que han solicitat llegir abans acabin de fer-ho per llegir ells.

2.2 Codi

```
void workload2(){

    int pid1,pid2,f,r,i;
    char buffer[32];
    pid = fork();

    if(pid1 == 0){
        pid2 = fork();
        if(pid2 == 0){
            r = read(0,&buffer, 150);
            prueba_estres(300000);
            r = read(0,&buffer, 150);
            print_status();
            exit();
        }
        else{
            r = read(0,&buffer, 150);
            prueba_estres(300000);
            r = read(0,&buffer, 150);
            print_status();
            exit();
        }
    }
    else{
        r = read(0,&buffer,1);
        r = read(0,&buffer, 150);
        prueba_estres(300000);
        r = read(0,&buffer, 150);
        print_status();
    }
}
```

3 Workload 3

3.1 Descripcio del proces

En aquest Workload els hi ha un process amb molta carrega de calcul i sense entrada sortida (el proces pare) mentre que els altres fan una barreja de Entrada/Sortida i calcul.

Round Robin

Amb aquesta política de planíficacio es remarca que processos que ocupen poc la CPU (els fills) acaben la seva execucio molt abans que el pare, ja que aquest va entrant i sortint de la CPU quan se li acaba el quantum i aixi permet que els recursos de la maquina siguin utilitzats eficientment per tots els processos.

First Come First Served

En aquest cas es pot observar clarament una millora del rendiment respecte a RR, ja que els processos fills han d'esperar a que l'execucio del pare.

Aquest Workload es una mostra de com una bona politica de planíficacio pot augmentar el rendiment del SO, ja que per exemple, si els processos que han de llegir es bloquejen durant l'execucio del pare (que no fa entrada/sortida) no han d'esperar a que el pare acabi per utilitzar recursos que estan disponibles.

3.2 Codi

```
void workload3(){

    char buff[32];
    int pid,pid_f,f,r;
    pid = fork();
    if(pid == 0){
        pid_f = fork();
        if(pid_f == 0){
            f = read(0, &buff,50);
            prueba_estres(600000);
            print_status();
            exit();
        }
        else{
            prueba_estres(1000000);
            r = read(0,&buff,200);
            prueba_estres(2000000);
            r = read(0,&buff,100);
            print_status();
            exit();
        }
    }
    else{
        prueba_estres(100000000);
        prueba_estres(100000000);
        print_status();
        exit();
}
}
```