

Evaluació de rendiment de polítiques de planificació de processos

Sergi Alonso, Pablo Llueca

8 de maig de 2014

1 Workload 1

1.1 Descripció del procés

El primer workload prova el rendiment de les diferents polítiques de planificació amb processos intensius en càlcul, amb molt poca entrada/sortida. El procés pare crea un fill i, que també crea un fill (3 processos en total) i tots executen un gran nombre (igual per a tots) d'iteracions d'un bucle que no fa res (funció `foo()`).

Round Robin

Aquesta política de planificació 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 procés idle no entra en cap moment en execució donat que mai estan tots els processos bloquejats.

First Come First Served

Amb aquesta política de planificació els processos fills només abandonen la CPU quan acaben la seva execució, i tot i que això no afecta al temps total sí que fa variar el temps que un procés individual passa a la cua de ready (el primer fill no està parat en cap moment, però el segon ha d'esperar que aquest acabi i el pare ha d'esperar que acabin els 2).

1.2 Codi

```
void foo(int n)
{
    int i, a;
    a = 23;
    for (i = 0 ; i < n; ++i)
        a = a + 1 - 4*i;
}
```

```

void workload1() {
    int pid, pid_f, f;
    pid = fork();
    if (pid == 0) {
        pid_f = fork();
        if (pid_f == 0) {
            foo(200000000);
            print_act(); exit();
        }
        else {
            foo(200000000);
            print_act(); exit();
        }
    }
    else {
        read(0, &buffer, 2);
        foo(200000000);

        print_act();
    }
}

```

2 Workload 2

2.1 Descripció del procés

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

Round Robin

En aquest cas la política de planificació casi ni actua, ja que cap procés executa càlculs llargs i per tant mai s'expulsa un procés de la CPU per que se li ha acabat el quantum.

First Come First Served

El resultat és molt similar a l'obtingut en RR, ja que tots els processos estan fortament lligats a l'entrada sortida, i es produeix una condició de carrera per accedir al dispositiu del qual es llegeix, per tant els processos hauran d'esperar a que els que han sol·licitat llegir abans acabin de fer-ho per llegir ells.

Es remarcable veure com el nombre de ticks que els processos que passen bloquejats s'acumula, ja que només hi ha un dispositiu del qual es llegeix (tot i que sigui virtual). Mentre els altres processos estan bloquejats el procés idle va executant-se.

2.2 Codi

```
void workload2() {
    int pid, pid_f, f, r, i;
    char buff[32];
    pid = fork();
    if (pid == 0) {
        pid_f = fork();
        if (pid_f == 0) {
            r = read(0, &buff, 1500);
            foo(300000);
            r = read(0, &buff, 1500);
            print_act();
            exit();
        }
        else {
            r = read(0, &buff, 1500);
            foo(300000);
            r = read(0, &buff, 1500);
            print_act();
            exit();
        }
    }
}
```

```
    }  
  }  
  else {  
    r = read(0,&buff,1);  
    r = read(0,&buff, 1500);  
    foo(300000);  
    r = read(0,&buff, 1500);  
    print_act();  
  }  
}
```

3 Workload 3

3.1 Descripció del procés

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

Round Robin

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

First Come First Served

En aquest cas es pot observar clarament una disminució del rendiment respecte a RR, ja que els processos fills han d'esperar a que l'execució del pare (que es substancialment mes llarga) acabi, amb el que pasen casi tot el seu temps de vida a la cua de ready.

Aquest Workload es una mostra de com una bona politica de planificació pot augmentar el rendiment del SO, ja que per exemple, si els processos que han de llegir es bloquejen durant l'execució 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;
    pid = fork();
    if(pid == 0){
        pid_f = fork();
        if(pid_f == 0){
            f = read(0, &buff, 50);
            foo(600000);
            print_act();
            exit();
        }
        else{
            foo(1000000);
            r = read(0, &buff, 200);
            foo(2000000);
            r = read(0, &buff, 100);
            print_act();
        }
    }
}
```

```
        exit ();
    }
}
else {
    foo(1000000000);
    foo(1000000000);
    print_act ();
    exit ();
}
}
```