ESPRIT

APPLICATION D'UN ALGORITHME APPROCHÉ POUR RÉSOUDRE UN PROBLÈME D'OPTIMISATION

Houssayen Ben Ouhiba

CE QUE NOUS ALLONS DISCUTER

PLAN

1)Donner le pseudo-code de chaque algorithme approché.

2) Expliquer l'utilité des différents paramètre de l'algorithme.

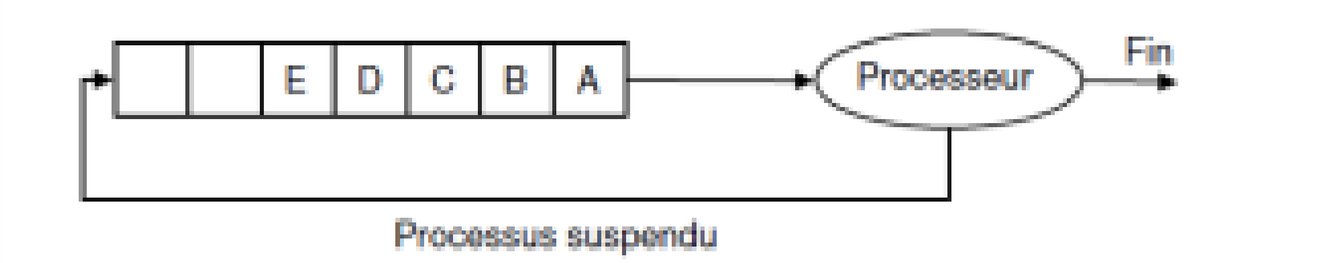
3)Comment peut-on appliquer chaque algorithme approché pour résoudre votre problème d'optimisation?

ORDONNANCEMENT CIRCULAIRE
ROUND ROBIN (RR)

- L'ALGORITHME DU TOURNIQUET, CIRCULAIRE OU ROUND ROBIN EST UN ALGORITHME ANCIEN, SIMPLE, FIABLE ET TRÈS UTILISÉ.

- IL MÉMORISE DANS UNE FILE DU TYPE FIFO (FIRST IN FIRST OUT) LA LISTE DES PROCESSUS PRÊTS, C'EST-À-DIRE EN ATTENTE D'EXÉCUTION

- ORDONNANCEURS PRÉEMPTIFS



```python
def findWaitingTime(processes, n, bt,
                    wt, quantum):
    rem_bt = [0] * n
    for i in range(n):
        rem_bt[i] = bt[i]
    t = 0 # Current time
    while(1):
        done = True
        for i in range(n):
            if (rem_bt[i] > 0) :
                done = False # There is a pending process
                if (rem_bt[i] > quantum) :
                    t += quantum
                    rem_bt[i] -= quantum
                else:
                    t = t + rem_bt[i]
                    wt[i] = t - bt[i]
                    rem_bt[i] = 0
        if (done == True):
            break
def findTurnAroundTime(processes, n, bt, wt, tat):
    for i in range(n):
        tat[i] = bt[i] + wt[i]
def findavgTime(processes, n, bt, quantum):
    wt = [0] * n
    tat = [0] * n
    findWaitingTime(processes, n, bt,
                    wt, quantum)
    findTurnAroundTime(processes, n, bt,
                                  wt, tat)
    print("Processes Burst Time  Waiting",
                   "Time Turn-Around Time")
    total_wt = 0
    total_tat = 0
    for i in range(n):

        total_wt = total_wt + wt[i]
        total_tat = total_tat + tat[i]
        print(" ", i + 1, "\t\t", bt[i],
            "\t\t", wt[i], "\t\t", tat[i])

    print("\nAverage waiting time = %.5f "%(total_wt /n) )
    print("Average turn around time = %.5f "% (total_tat / n))
if __name__ =="__main__":
    proc = [1, 2, 3]
    n = 3
    burst_time = [10, 5, 8]
    quantum = 2;
    findavgTime(proc, n, burst_time, quantum)
```

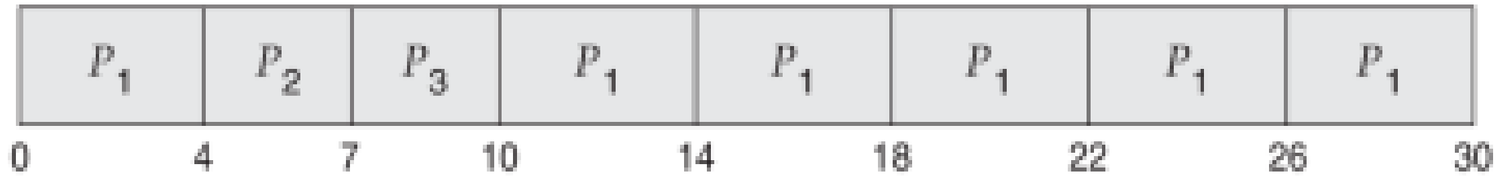
PSEUDO-CODE:

PYTHON

Exemple de RR

P1	24
P2	3
P3	3

- Quantum = 4ms



L'UTILITÉ DES DIFFÉRENTS PARAMÈTRE DE L'ALGORITHME

```
// Method to calculate average time
static void findavgTime(int processes[], int n, int bt[],
                                    int quantum)
{
    int wt[] = new int[n], tat[] = new int[n];
    int total_wt = 0, total_tat = 0;
```

=>Methode pour calculer average duree

```
// Function to find waiting time of all processes
findWaitingTime(processes, n, bt, wt, quantum);
```

=>Methode pour trouver le temps d'attente de chaque processus

```
// Function to find turn around time for all processes
findTurnAroundTime(processes, n, bt, wt, tat);
```

=>Methode pour trouverdélai d'exécution de tous les processus

```
// Time quantum
int quantum = 2;
findavgTime(processes, n, burst_time, quantum);
```

- **Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur.**
- **Un quantum trop élevé augmente le temps de réponse des courtes commandes en mode interactif.**
- **Un quantum entre 20 et 50 ms est souvent un compromis raisonnable**

```
int bt[],
```

BURST TIMES.

```
int wt[],
```

WAITING TIME.

```
int tat[
```

TURN AROUND TIME (DÉLAI D'EXÉCUTION

ON PEUT APPLIQUER NOTRE ALGORITHME APPROCHÉ POUR RÉSOUDRE NOTRE PROBLÈME D'OPTIMISATION

- Round Robin est un algorithme de planification de CPU où chaque processus se voit attribuer un intervalle de temps fixe de manière cyclique. C'est simple, facile à mettre en œuvre et sans famine car tous les processus obtiennent une part équitable de l'UC. L'une des techniques les plus couramment utilisées dans la planification du processeur en tant que noyau.

- le round robin est meilleur pour les travaux courts, mais il est mauvais pour les travaux de même durée.