Modèle de base :

- Nombre de tâches fixe
- Tâches périodiques et indépendantes (pas de partage de ressources, synchronisation, communication)
- Temps de commutation négligeable

d. drapo

Ordonnancement cyclique : A, B, C, D

```
while (true) {
A();
B();
C();
attendre interruption;
}
```

23/11/2022

Algorithmes classiques :

- FIFO
- plus courte durée en premier (SJF)
- circulaire (round robin ou tourniquet)

Exemple d'évaluation d'algorithmes / temps moyen d'attente :

↳ Considérant le pire cas de figure : lancement de toutes les tâches simultanément à $t=0$

Tâches	A	B	C	D	E
Temps d'exécution (µs)	10	29	3	7	12

- **En FIFO,** le temps d'attente est (date de début - date lancement)

0 µs pour A
10 µs pour B
39 µs pour C
42 µs pour D
49 µs pour E

Temps d'attente moyen $M = \frac{0+10+39+42+49}{5} = 28\ \mu s$

Tâche	A	B	C	D	E
Temps de réponse (µs)	10	39	42	49	61

=> Temps de réponse moyen = 40,2 µs

. En SJF (Short Job First)

Tâche	A	B	C	D	E	Moyenne
Temps d'attente (µs)	10	32	0	3	20	13
Temps de réponse (µs)	20	61	3	10	32	25,2

=> amélioration des performances

Temps de réponse (date fin d'exécution - date d'activation)

Tourniquet avec Z = 10 µs (période de préemption

Tâche	A	B	C	D	E	Moyenne
Temps d'attente (µs)	0	61-29 = 32	20	23	40	23
Temps de réponse (µs)	10	61	23	30	52	35,2

=> cet algorithme n'améliore pas les performance => prévoir en plus des temps de changement de contexte

⚠ temps d'attente ne prend pas en compte le temps d'exécution
=> à décompter dans le cas d'une exécution en plusieurs fois.

En conclusion, l'algorithme SJF donne les meilleurs performances

Ordonnancement selon la période : Rate Monotonic Algorithm (RMA) ou Rate Monotonic Scheduler (RMS).

-> algorithme en ligne à priorité constante

-> tâches doivent être indépendantes et pouvant s'exécuter dans n'importe quel ordre

-> priorité inversement proportionnelle à la période

Condition suffisante $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \xrightarrow[n \to +\infty]{} 0{,}69$

cf. dispo

-> optimal pour des tâche à échéances sur requête

Dans le cas d'incertitude où la condition suffisante n'est pas vérifiée

Inverse Deadline ou Deadline Monotonic. la tâche la plus prioritaire est celle du plus petit délais critique.

Performances : équivalente à RMA par des tâches à échéance sur requête.

Condition suffisante d'ordonnançabilité $U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$

cf. diapo

Critère d'ordonnançabilité $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ = (ici 0,8)

cf. diapo

Earliest Deadline First (EDF)

Condition suffisante d'ordonnançabilité $U = \sum_{i}^{n} \frac{C_i}{D_i} \leq 1$

Condition nécessaire et suffisante ($T_i = D_i$) $U = \sum_{i}^{n} \frac{C_i}{T_i} \leq 1$

Calcul du temps de réponse maximum : Worst Case Response Time (WCRT)

Pour des tâches à priorité fixe et indépendantes.

cf diapo

$$T_{r_i} = C_i + \sum_{j \in hp(i)} \text{Attente\_tâche } j.$$

$$T_{r_i}^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{T_{r_i}^{n}}{T_j} \right\rceil \times C_j$$

↑ borne supérieure

avec $hp(i)$: Ensemble des tâches de priorité supérieure à celle de la tâche i

Exemple : $T_{r_3}^{0} = 2$

$$T_{r_3}^{1} = 2 + \frac{2}{5} + 2 \simeq 4.$$

↳ unité de partie entière.

$$T_{r_2}^{0} = 4$$

$$T_{r_2} = C_1 + \left\lceil \frac{T_{r_2}^{0}}{T_2} \right\rceil C_2 + \left\lceil \frac{T_{r_2}^{1}}{T_3} \right\rceil C_3$$

$$= 4 + \left\lceil \frac{4}{5} \right\rceil \times 2 + \left\lceil \frac{4}{10} \right\rceil \times 2$$

$$Tr_2^2 = C_1 + \left\lceil \frac{Tr_2^1}{T_2} \right\rceil C_2 + \left\lceil \frac{Tr_1^1}{13} \right\rceil C_3$$

itère jusqu'à ce que le résultat égale l'itération précédente

$$= 4 + \left\lceil \frac{8}{5} \right\rceil \times 2 + \left\lceil \frac{8}{10} \right\rceil \times 2 = 10$$

$$Tr_2^3 = C_1 + \left\lceil \frac{Tr_2^2}{T_2} \right\rceil \times C_2 + \left\lceil \frac{Tr_2^2}{T_3} \right\rceil \times 3$$

$$= 4 + \left\lceil \frac{10}{5} \right\rceil \times 2 + \left\lceil \frac{10}{10} \right\rceil \times 2 = 10$$

$\Rightarrow Tr_2^2 = 10$ correspond au $WCRT_2$

$\Rightarrow WCRT_2 = C_2 = 2$ et

$Tr_3^0 = C_3 = 2 \Rightarrow WCRT_3 = 4$

Tâches partageant des ressources :

Durée de blocage avec protocole d'héritage de priorité :

$$B_i = \sum_{k=2}^{K} usage(k,i)\, C(k)$$

Durée de blocage avec OPCP ou IPCP :

$$B_i = \max_{k=2}^{K} usage(k,i)\, C(k)$$

pour K : nombres de ressources

$$Usage(k,i) = \begin{cases} 1 \text{ si } \exists T_j \text{ utilise } k \text{ et } Prio(T_j) < Prio(T_i) \\ \exists T_n \mid T_n \text{ utilise } k \text{ et } Prio(T_n) \geq Prio(T_i) \text{ incluant } T_i \\ 0 \text{ sinon} \end{cases}$$

$C(k)$ durée d'utilisation maximum de la ressource k par une tâche T_j

Worst Case Execution Time (WCET) : méthodologie basée sur des jeux de données d'entrées :
-> Déterminer la combinaison des entrées la plus pessimiste
-> Exécuter le code avec le maximum de combinaison d'entrées possible et mesurer le temps d'exécution.
-> Ajouter une marge de sécurité

En pratique : . Réaliser une approche statistique (garantir 6 sigma i.e. dans 99,999 % des cas en aéronautique)
. Mesurer pour en tirer une estimation

Méthode à base de Plots :

Remarque : On tend à avoir des algo de complexité $O(n)$

⚠ influence du matériel :
Un processeur moderne (RISC) classique implique un pipeline à 5 niveaux

int