IT-201 TD2-correction

baptiste.coye

February 2020

1 Implémentation de base

2-a

```
void wait()
{
  /* on s'ajoute au debut de la sleepq */
  current->p_next = sleepq;
  sleepq = current;
  /* on se marque en dodo sur l'evenement */
  current->p_stat = STATUS_BLOCKED;
  /* on rend la main */
  switch();
}

void wakeup()
{
  while (sleepq) {
    /* on enleve de la sleepq */
    struct proc * myproc = sleepq;
    sleepq = myproc->p_next;
    myproc->p_next = NULL;
    /* on marque reveill  */
    myproc->p_stat = STATUS_READY;
    /* on met sur la runq */
    myproc->p_next = runq;
    runq = myproc;
  }
  sleepq = NULL;
}
```

Dans ce cas là le status n'est pas strictement nécessaire, il est équivalent à être sur la Run Queue ou la Sleep Queue. Cependant il est utile pour le switch() pour savoir si on veut s'endormir ou si l'ordonnanceur peut nous rendre la main

si aucuns autre processus ne la demande.

2-b

Une interruption du disque peut techniquement reveiller un processus qui attend une lecture disque. Cela fait qu'un wakeup peut être appelé au milieu d'un wait, par exemple entre les modifications de la SleepQ.
Sur le même principe, un wakeup peut également être appelé au milieu d'un autre wakeup si une interruption matérielle (irq) arrive quand un autre processus fait un wakeup (Exemple : Ecriture dans un tube sur lequel un processus attend en lecture). Pour prévenir cela on protége wait et wakeup des irqs.

```
void wait()
{
  irq_disable();
  current->p_stat = STATUS_BLOCKED;
  /* on s'ajoute au debut de la sleepq */
  current->p_next = sleepq;
  sleepq = current;
  irq_enable();
  /* on rend la main */
  switch();
}

void wakeup()
{
  irq_disable();
  while (sleepq) {
    /* on enleve de la sleepq */
    struct proc * myproc = sleepq;
    sleepq = myproc->p_next;
    myproc->p_next = NULL;
    /* on marque reveill  */
    myproc->p_stat = STATUS_READY;
    /* on met sur la runq */
    myproc->p_next = runq;
    runq = myproc;
  }
  sleepq = NULL;
  irq_enable();
}
```

Notes: On peut remarquer qu'il vaut mieux garder les irqs autour du changement de status car un switch alors que le status est BLOCKED sans être dans la file peut rendre la main sans jamais la reprendre si l'ordonnanceur verifie le status quand il va piocher dans la RunQ.

Pour un exemple d'implémentation réelle, se référer au code de linux qui permet l'attente ou le wakeup.

- wait_for_completion : https://git.kernel.org/pub/scm/linux/kernel/git/wtarreau/linux-2.4.git/tree/kernel/sched.c?h=v2.4.37n768
- try_to_wake_up : https://git.kernel.org/pub/scm/linux/kernel/git/wtarreau/linux-2.4.git/tree/kernel/sched.c?h=v2.4.37n349

2-c

Dans le cas ou un processus veuille s'endormir et qu'avant qu'il rende la main une interruption réveil tous les processus en attente, il va se placer dans la file d'attente puis être reveillé immediatement avant switch et rendre la main alors que ce n'était pas nécessaire. Dans ce cas là il perd du temps inutilement. Cela ne pose aucuns problèmes mais entraîne une perte de performance. Pour éviter que cela se produise, on pourrait changer l'implémentation du switch afin de vérifier que le process rendant la main a bien un STATUS_BLOCKED

2-d

Dans le cas où notre machine était monoprocesseur on ne pouvait avoir qu'un processus dans wait ou wakeup interrompu par un wakeup d'une interruption matérielle. En multiprocesseurs, on a alors plusieurs processus dans wait ou wakeup en même temps (même sans interruptions). Dans ce cas là il faut protéger la SleepQ et la RunQ grâce à des verrous.
On utilise normalement des verrou en attente active (spin_lock dans linux), pas un verrou qui endort le processus qui attend. Il faut tout de même désactiver les interruptions avant de prendre le verrou, de cette façon nous pouvons opérer sur la RunQ et la SleepQ en sécurité puis relacher le verrou avant de réactiver les interruptions.

2 Réception de paquets réseau

3-a

Il est nécessaire dans la fonction wakeup d'incrémenter la valeur de received_count mais il faut alors désactiver les interruptions pour empêcher d'avoir des accès concurents à cette variable.

```
void wakeup()
{
  irq_disable();

  received_count++;

  while (sleepq) {
    struct proc * myproc = sleepq;
    sleepq = myproc->p_next;
    myproc->p_stat = STATUS_READY;
    myproc->p_next = runq;
    runq = myproc;
  }
  irq_enable();
}
```

Dans le wait, il faut vérifier le received_count avant de dormir. Il est possible d'autoriser les interruptions avant le consume car on est sûr qu'il y aura au moins un paquet disponible à consommer. De plus on gagne en performence car cela évite de bloquer la machine si la fonction consume n'est pas assez rapide.

```
void wait()
{
  while (1) {
    irq_disable();
    if (received_count) {
      /* c'est bon, il y a un paquet pour nous, on marque qu'on le prend */
      received_count--;
      irq_enable();
      /* maintenant on peut le prendre en securit  */
      consume();
      return;
    }
    current->p_stat = STATUS_BLOCKED;
    current->p_next = sleepq;
    sleepq = current;
    irq_enable();
    switch();
    /* quelqu'un nous a reveill  , mais verifions qu'il reste bien un paquet pou
  }
}
```