Projet 4IF : Raspberry PI

H4104 - équipe Dragibus

INSA de Lyon

Vendredi 13 décembre 2013

Introduction

Sommaire

Synchronisation des processus

conception en couches

Synchronisation des processus

conception en couches

1. **sémaphore** : compteur à intervalle bloquant le processus appelant (opération +1/-1) s'il cause un dépassement

Synchronisation des processus

conception en couches

1. **sémaphore** : compteur à intervalle bloquant le processus appelant (opération +1/-1) s'il cause un dépassement
2. **mutex** : sémaphore de 0 à 1, permettant un acquire/release, uniquement par un processus à la fois

Synchronisation des processus

conception en couches

1. **sémaphore** : compteur à intervalle bloquant le processus appelant (opération +1/-1) s'il cause un dépassement
2. **mutex** : sémaphore de 0 à 1, permettant un acquire/release, uniquement par un processus à la fois
3. **pipe** : FIFO avec mutex associé

Problème de synchronisation (résolu)

```
void sem_op(struct sem_s * sem, struct sem_op_s * op) {
    DISABLE_IRQ();

    /* operation sur le semaphore */

    ENABLE_IRQ();
}
```

Listing 1 – Opération atomique sur un sémaphore

Les problématiques d'ordonnancement

Objectifs

Les problématiques d'ordonnancement

Objectifs

- rapide

Les problématiques d'ordonnancement

Objectifs

- rapide
- automatique

Les problématiques d'ordonnancement

Objectifs

- rapide
- automatique
- équitable

Ordonnancements mis en place

Ordonnancements mis en place

- préemptif

Ordonnancements mis en place

- préemptif
- collaboratif

Ordonnancements mis en place

- préemptif
- collaboratif
- prioritaire

Critique des différents ordonnancements

- préemptif : aucune distinction entre les différents processus
- collaboratif : manque d'automatisme (**yield()** nécessaire)
- prioritaire : problématique du *bon* ordre

Politique d'ordonnancement choisie

combinaison

Ordonnancement d'une tâche

```
struct task {
    /* Etat de la tache */
    int state, needs_reschedule;

    /* Politique d'ordonnancement */
    int policy;

    /* Priorite */
    time_t priority, realtime_priority,
        current_priority;

    /* Processus suivant/precedent */
    struct task * next, * prev;

    /* contexte, informations temporelles, etc... */
};
```

Listing 2 – Représentation structurelle d'un processus

Ordonnancement d'une tâche

Performances du mini-OS

Problématiques d'ordonnancement sous Linux

Exemple : short vs long

Concurrence entre deux processus ayant des politiques d'ordonnancement différentes

Exemple : short vs long

```
void run_long_process {
    sched_process(NORMAL, LONG_PRIORITY);
    for (size_t i = 0; i < LONG_NB_COMPUTATIONS; i++) {
        printf_safe("LONG\n");
        /* calcul long */
    }
}
```

Listing 3 – processus long

Exemple : short vs long

```
void run_short_process {
    sched_process(REAL_TIME, SHORT_PRIORITY);
    for (;;) {
        printf_safe("SHORT\n");
        /* calcul court */
    }
}
```

Listing 4 – processus court

Exemple : short vs long

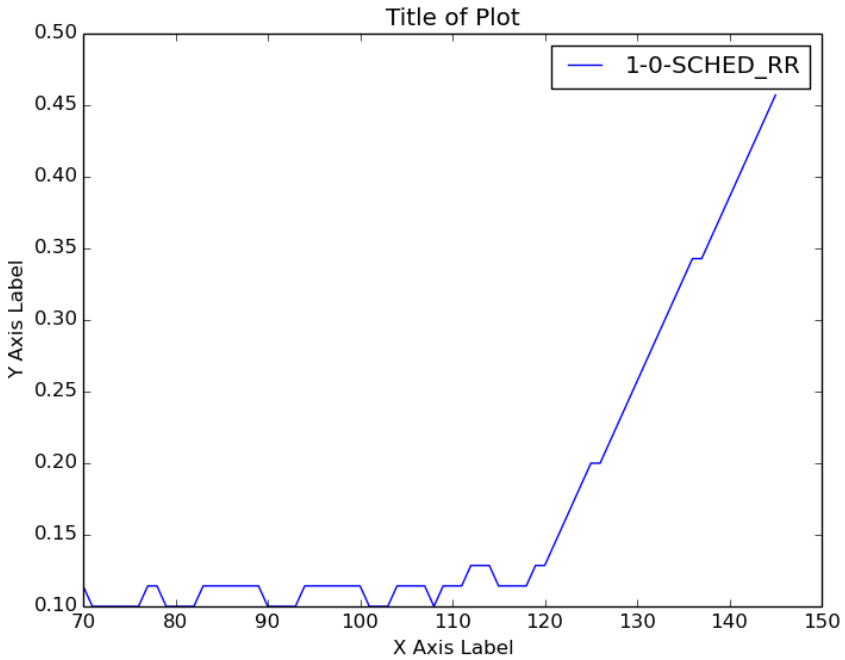

FIGURE : Court bas, long normal

Exemple : short vs long

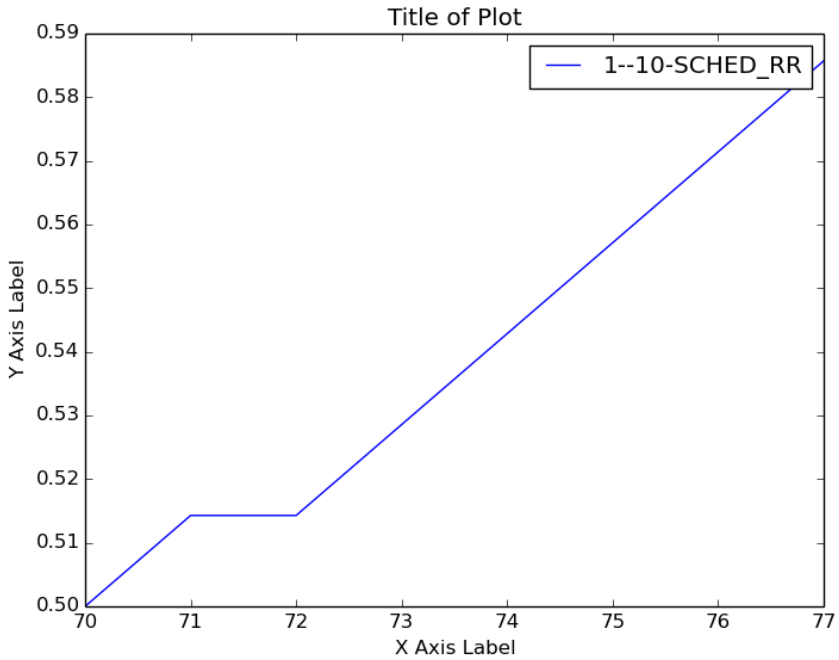

FIGURE : Court bas, long haut

Exemple : short vs long

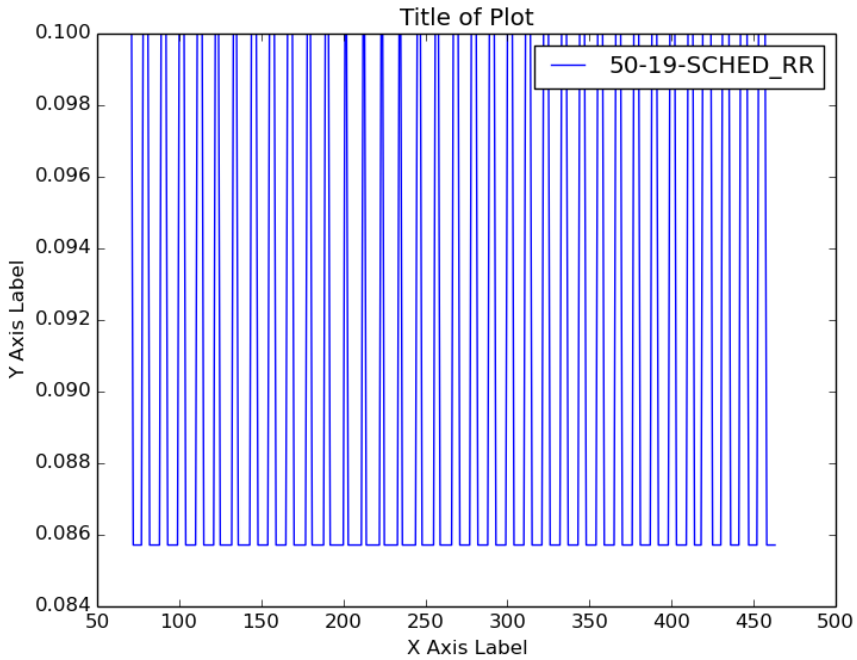

FIGURE : Court moyen, long bas

Exemple : short vs long

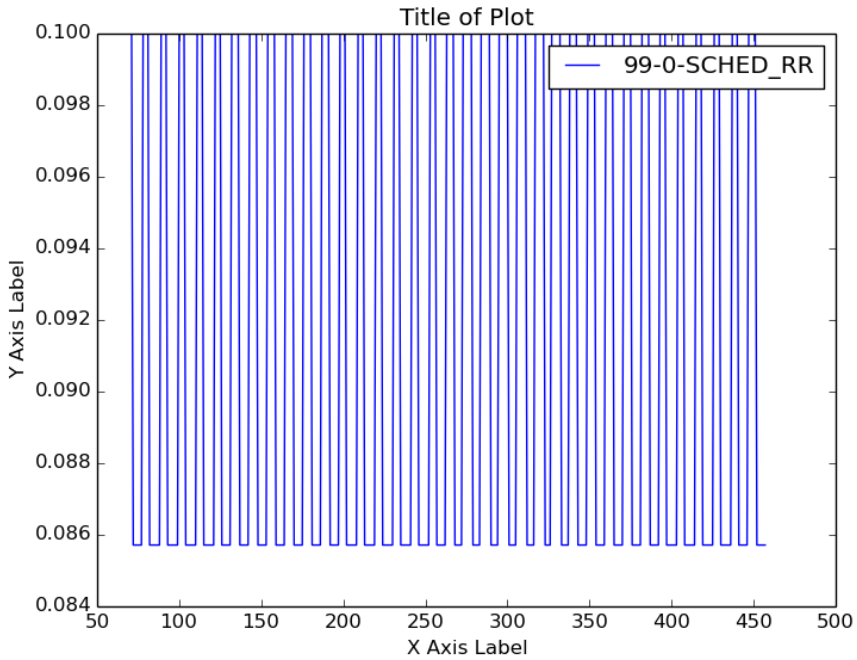

FIGURE : Court haut, long moyen

Compromis CPU / Requêtes IO

Analyse des algorithmes

```
struct task * choose_next_task() {
    struct task * next = NULL;
    for (unsigned int i = 2*MAX_PRIO - 1 ; i >= 0 ; i--) {
        if (realtime_active_tasks->queue[i]) {
            return realtime_active_tasks->queue[i];
        } else if (!next && active_tasks->queue[i]) {
            next = active_tasks->queue[i];
        }
    }
    return next;
}
```

Listing 5 – Choix de la prochaine tâche à exécuter

Analyse des algorithmes

```
void detect_task_activity() {
    struct task_struct * prev = current_task;
    if (prev != &idle_task && prev->state == TASK_READY &&
         prev->policy & SCHED_YIELD) {
        size_t k = prev->current_priority;
        active_tasks->queue[k] =
            queue_del(active_tasks->queue[k], prev);
        active_tasks->queue[0] =
            queue_push(active_tasks->queue[0], prev);
        prev->policy &= ~SCHED_YIELD;
        prev->need_reschedule = 0;
        prev->current_priority = 0;
    }
}
```

Listing 6 – Détection de l'activité d'une tâche (IO/CPU)

Problèmes divers rencontrés

Problème

Aucune sortie visible n'est disponible pour déboguer les fonctionnalités du kernel.

Problèmes divers rencontrés

Problème

Aucune sortie visible n'est disponible pour déboguer les fonctionnalités du kernel.

Solution

Nous traduisons les messages par un clignotement de la LED disponible, suivant un **code morse**.