Rapport Exercice 6 – TP0

Belmouloud Mustapha Abdellah — Groupe 1

Introduction

L'énoncé demande de créer trois processus enfants : l'un pour compter les secondes, l'autre pour les minutes et le dernier pour les heures. Le principal problème est donc de trouver un moyen de communication entre ces processus enfants afin d'afficher le temps correctement.

Solution de l'exercice

Pour résoudre ce problème, on utilise **sigqueue** et **sigaction** afin d'envoyer des signaux **SIGUSR1** contenant des données (le PID de l'émetteur et des variables). Ces signaux reposent sur des structures prédéfinies en C. Cela permet une communication hiérarchique et fiable entre les processus.

Explication de la syntaxe

La fonction **sigqueue()** permet d'envoyer un signal à un processus spécifique tout en ajoutant une valeur entière associée (via *union sigval*). Contrairement à **kill()**, les signaux sont mis en file d'attente, ce qui évite les pertes.

La fonction **sigaction()** est utilisée pour définir un gestionnaire de signaux avancé. En utilisant l'option **SA_SIGINFO**, on peut récupérer des informations supplémentaires comme le PID de l'émetteur ou la valeur envoyée (*info->si_value.sival_int*).

Syntaxe générale

```
#define _POSIX_C_SOURCE 200809L
#include <signal.h>
#include <unistd.h>
```

Envoi d'un signal avec une valeur

```
union sigval sv;
sv.sival_int = value;
sigqueue(pid, SIGUSR1, sv);
```

Réception du signal

```
void handler(int sig, siginfo_t *info, void *context) {
    printf("Valeur reçue : %d\n", info->si_value.sival_int);
}

int main(void) {
    struct sigaction sa = {0};
    sa.sa_sigaction = handler;
    sa.sa_flags = SA_SIGINFO;
    sigaction(SIGUSR1, &sa, NULL);
}
```

Principe de fonctionnement

Le père envoie les PID des processus Heures (H) et Minutes (M) aux processus Minutes (M) et Secondes (S). Chaque processus enfant s'exécute dans une boucle :

- **Secondes (S)** : incrémente chaque seconde et envoie la valeur au père. Si *seconds* == *60*, il remet à zéro et envoie un signal à M.
- **Minutes (M)** : incrémente à chaque signal de S, et si *minutes* == *60*, il remet à zéro et envoie un signal à H.
- **Heures (H)** : incrémente à chaque signal de M et renvoie la valeur au père.

La fonction **pause()** bloque chaque processus en attendant un signal, et la variable **ready** garantit que tous les PID ont été échangés avant le démarrage du programme.

Extrait : Processus Secondes (S)

```
S = fork();
if (S == 0) {
    struct sigaction sa = {0};
    sa.sa_sigaction = Srec;
    sa.sa_flags = SA_SIGINFO;
    sigaction(SIGUSR1, &sa, NULL);

    while (!ready) pause();

    while (true) {
        sleep(1);
        seconds++;
        union sigval svv = { .sival_int = seconds };
        sigqueue(F, SIGUSR1, svv);

        if (seconds == 60) {
            seconds = 0;
            union sigval sv = { .sival_int = 0 };
            sigqueue(M, SIGUSR1, sv);
        }
    }
    exit(0);
}
```

Extrait : Processus Minutes (M)

```
M = fork();
if (M == 0) {
    struct sigaction sa = {0};
    sa.sa_sigaction = Mrec;
    sa.sa_flags = SA_SIGINFO;
    sigaction(SIGUSR1, &sa, NULL);

    while (!ready) pause();

    while (true) {
        pause();
        minutes++;
        union sigval svv = { .sival_int = minutes };
        sigqueue(F, SIGUSR1, svv);
```

```
        if (minutes == 60) {
            minutes = 0;
            union sigval sv = { .sival_int = 0 };
            sigqueue(H, SIGUSR1, sv);
        }
    }
    exit(0);
}
```

Extrait : Processus Heures (H)

```
H = fork();
if (H == 0) {
    struct sigaction sa = {0};
    sa.sa_sigaction = Hrec;
    sa.sa_flags = SA_SIGINFO;
    sigaction(SIGUSR1, &sa, NULL);

    while (true) {
        pause();
        hours++;
        union sigval svv = { .sival_int = hours };
        sigqueue(F, SIGUSR1, svv);
    }
    exit(0);
}
```

Modification : fonction Mrec

Pour un bon résultat, il faut modifier la fonction de M qui reçoit les signaux afin de distinguer entre les signaux du père et ceux du processus S :

```
void Mrec(int sig, siginfo_t *info, void *context) {
    if (info->si_pid == F) {
        H = info->si_value.sival_int;
        ready = true;
    }
    if (info->si_pid == S) {
        // ...
    }
}
```

Fonction du père : affichage des résultats

Et finalement, la fonction du père qui affiche les résultats doit utiliser **fflush(stdout)** pour garantir l'affichage immédiat. Elle doit aussi vérifier si les secondes ou les minutes sont égales à 60 pour éviter les doublons. De plus, le signal des minutes ne doit être déclenché que lorsque les secondes reviennent à zéro (c'est-à-dire lorsque *seconds == 0*), et la même logique s'applique pour les heures et les minutes.

```
void Frec(int sig, siginfo_t *info, void *context) {
    if (info->si_pid == S) {
        seconds = info->si_value.sival_int;
        if (seconds != 60)
            printf("%02d : %02d : %02d\n", hours, minutes, seconds);
    }
    if (info->si_pid == M) {
        minutes = info->si_value.sival_int;
```

```
        if (minutes != 60)
            printf("%02d : %02d : %02d\n", hours, minutes, 0);
    }
    if (info->si_pid == H) {
        hours = info->si_value.sival_int;
        printf("%02d : %02d : %02d\n", hours, 0, 0);
    }
    fflush(stdout);
}
```

Conclusion

Les méthodes classiques comme **kill()** et **signal()** ne permettent pas un contrôle précis sur la transmission des données entre processus. L'utilisation combinée de **sigqueue()** et **sigaction()** résout ce problème en assurant la fiabilité, le transport de valeurs et la mise en file d'attente des signaux. Ce modèle peut être adapté à d'autres systèmes nécessitant une synchronisation entre plusieurs processus. En alternative, l'utilisation de **pipelines** permettrait une communication continue entre processus, mais sans la légèreté et la précision des signaux POSIX.