IMA 4 - T.P. Système
Ordonnancement

1 Objectifs

L'objectif de ce TP consiste à réaliser un ordonnanceur de tâches de type *Round Robin*. Pour simplifier le développement et mieux appréhender le concept de tâche, cet ordonnanceur fonctionnera sur une plate-forme Arduino. Elle est architecturée autour d'un microcontrôleur de la famille AVR d'Atmel (`atmega328p`).

Dans le cadre du TP, le développement sur cette plate-forme se fera en utilisant majoritairement le langage C. Quelques instructions assembleur nécessaires pour la réalisation de l'ordonnanceur seront indiquées dans le sujet.

Pour vous simplifier le travail, vous devez récupérer le fichier `http://tvantroys.plil.fr/enseignement/IMA4_OS/scheduler/scheduler.tgz` et le décompresser dans votre compte.

2 Prise en main de la plate-forme

Afin de découvrir le développement sur la plate-forme Arduino, vous allez dans un premier temps réaliser un programme qui fera uniquement clignoter une LED.

Placez vous dans le répertoire `exo1` et à l'aide de `vim` éditez le fichier `main.c`. La fonction principale d'un programme d'un microcontrôleur s'articule autour d'une boucle infinie.

Pour allumer ou éteindre une LED, il faut configurer le registre d'entrée/sortie sur lequel est connecté la LED. Comme vous pouvez le voir en regardant le schematic (arduino nano `http://arduino.cc/en/uploads/Main/ArduinoNano30Schematic.pdf` ou arduino uno `http://arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf`) la LED rouge utilise l'entrée/sortie 0 du port B et la led jaune utilise l'entrée/sortie 1 du port B. Le registre de configuration se nomme `DDRB`. Chaque bit de ce registre représente une entrée/sortie du port B. Un 0 signifie une utilisation en tant qu'entrée et un 1 signifie une utilisation en tant que sortie.

Écrivez la fonction `init_led` afin de pouvoir utiliser les deux LEDS.

Pour activer une sortie, il faut positionner à 1 le bit correspondant du registre `PORTB`.

Écrivez la boucle principale afin d'allumer la LED rouge pendant 200 ms, l'éteindre puis allumer la LED jaune pendant 300 ms puis retour à l'allumage de la LED rouge. Afin de temporiser, il faut utiliser la fonction `_delay_ms` qui prend en paramètre le nombre de millisecondes à attendre.

Pour compiler, vous devez utiliser le Makefile fourni. La cible `all` permet de compiler et la cible `upload` permet de transférer le binaire dans la plate-forme Arduino. Pour utiliser la cible `upload` sur les machines de TP, il faut au préalable brancher la plate-forme Arduino via le câble USB et utiliser la commande `super usb` afin de modifier les droits sur le périphérique associé à l'arduino (`/dev/ttyUSB0` pour un arduino nano ou `/dev/ttyACM0` pour un arduino uno).

Vérifiez le bon fonctionnement de votre programme.

3 Exécution de deux tâches en parallèle : utilisation d'un timer

Vous allez maintenant utiliser l'un des timers du microcontrôleur pour faire clignoter les 2 LEDs en parallèle.

Placez-vous dans le répertoire `exo2` et commencez par écrire le code de la fonction `task_led_red` qui fait clignoter la LED rouge toute les 200 ms (i.e., la LED est allumée pendant 200 ms, puis éteinte pendant 200 ms). Cette fonction est appelée dans la boucle de la fonction `main`. Testez le bon fonctionnement.

Afin de faire clignoter la LED jaune toutes les 400 ms en parallèle avec la LED rouge (i.e., le LED est allumée pendant 400 ms, puis éteinte pendant 400 ms), nous allons utiliser un timer. À chaque cycle de l'horloge, un compteur est incrémenté et il est comparé à la valeur se trouvant dans un registre (registre `OCR1A` dans notre cas). Si ils sont égaux, une interruption est générée. Le programme est alors dérouté pour exécuter la fonction correspondante et le compteur est réinitialisé. Les fonctions d'interruptions, dans le cas d'un microcontrôleur AVR et de l'utilisation de avr-gcc sont nommées `ISR` et prennent en paramètre le vecteur d'interruption correspondant (`TIMER1_COMPA_vect` dans notre cas). Le timer est initialisé avec un *prescaler* de 1024, cela signifie que le compteur est incrémenté tous les 1024 cycles. L'arduino est cadencé à 16 MHz. Vous pouvez donc déterminer la valeur du nombre pour initialiser le registre `OCR1A`) afin que la fonction d'interruption soit appelée toutes les 400 ms. Cette fonction réalisera ainsi le changement d'état de la LED jaune. Pour activer les interruptions vous devez appeler la fonction `sei()`. Pour désactiver les interruptions, la fonction est `cli()`.

Réalisez et testez le programme.

Grâce à l'utilisation du timer, vous avez pu réaliser ainsi deux tâches qui s'exécutent en parallèle.

4 Ordonnancement

Vous allez maintenant réaliser un véritable ordonnanceur utilisant comme algorithme un *round robin* avec un intervalle de temps de 20 ms. Votre microcontrôleur réalisera trois tâches :

1. faire clignoter la LED rouge toutes les 300 ms
2. envoyer en boucle un message sur le port série (chaque envoi de caractère sera espacé de 100 ms)
3. envoyer en boucle un message sur une matrice de LED communiquant via le bus SPI

Placez vous dans le répertoire `exo3`. Le fichier `RGB.c` contient les différentes fonctions permettant d'utiliser la matrice de LEDs. Les deux fonctions que vous devrez utiliser sont `init_rgb()` et `rgb_main()`. Le fichier `main.c` contient les fonctions d'initialisation du port série et d'envoi d'un caractére sur le port série. Avant de réaliser l'ordonnanceur, écrivez le code de la tâche qui envoie un message sur le port série et testez-le. Faites de même avec la tâche qui envoie un message sur la matrice de LED.

Maintenant que vos différentes tâches fonctionnent, vous pouvez réaliser l'ordonnanceur. Vous devez commencer par créer une structure représentant les différentes tâches.

Comme vous l'avez lu dans l'article envoyé avant les vacances, pour réaliser le changement de contexte, il est nécessaire de sauvegarder la valeur des différents registres du microcontrôleur. Pour cela, vous allez compléter les deux macros `SAVE_CONTEXT` et `RESTORE_CONTEXT`. Pour la gestion de la mémoire des différents processus, nous allons découper l'espace mémoire. Dans un AVR atmega328p, la mémoire est utilisée en commençant par les adresses les plus élevées. La tâche "écriture série" débutera son utilisation mémoire à l'adresse 0x0700, la tâche "écriture matrice de LED" débutera à l'adresse 0x0600 et la tâche "LED" débutera à l'adresse 0x0500. Ces adresses représentent le départ du pointeur de pile (*Stack Pointer*). Ce dernier est rendu directement accessible par gcc-avr via la "variable" `SP`.

L'ordonnanceur utilise l'algorithme du tourniquet (*Round Robin*) avec un intervalle de 20 ms. Pour faciliter le debug, chaque fois que l'ordonnanceur s'exécute, vous allumerez la LED jaune.

5 Ressource partagée

On souhaite maintenant rajouter une tâche supplémentaire qui envoie en boucle le caractère "!" sur le port série. Ceci nécessite donc de gérer l'accès concurrent au port série entre deux tâches.

Vous allez mettre en place un mécanisme simplifié de sémaphores permettant d'assurer l'accès en exclusion mutuelle au port série. Pour cela :

1. Il faut désormais trois états possibles pour une tâche : `CREATED` (la tâche n'a encore jamais été exécutée), `ACTIVE` (la tâche est disponible) et `SUSPENDED` (la tâche attend l'accès à une ressource partagée).
2. Ecrivez une primitive `take_serial` (similaire au P/Wait des sémaphores), qui vérifie si le port série est disponible, si oui le prend, si non la tâche l'exécutant est suspendue.
3. Écrivez une primitive `release_serial` (similaire au V/Signal des sémaphores), qui rend le port série et, si besoin, rend active la tâche suspendue.
4. Modifiez le code des tâches et de l'ordonnanceur en conséquence.

Notez bien que les primitives `take_serial` et `release_serial` doivent être ininterruptibles !

6 Pour aller plus loin

Pour aller plus loin, reprenez votre code et remplacez le tourniquet par un algorithme âge et priorité. La priorité 0 est la plus élevée. L'intervalle de temps passe à 1 s. La tâche d'écriture sur le port série a la priorité 10, la tâche de la matrice de LED a la priorité 5 et la tâche de clignotement de la LED rouge a la priorité 2.