GIF-3002 Interfaces et Système d'exploitation

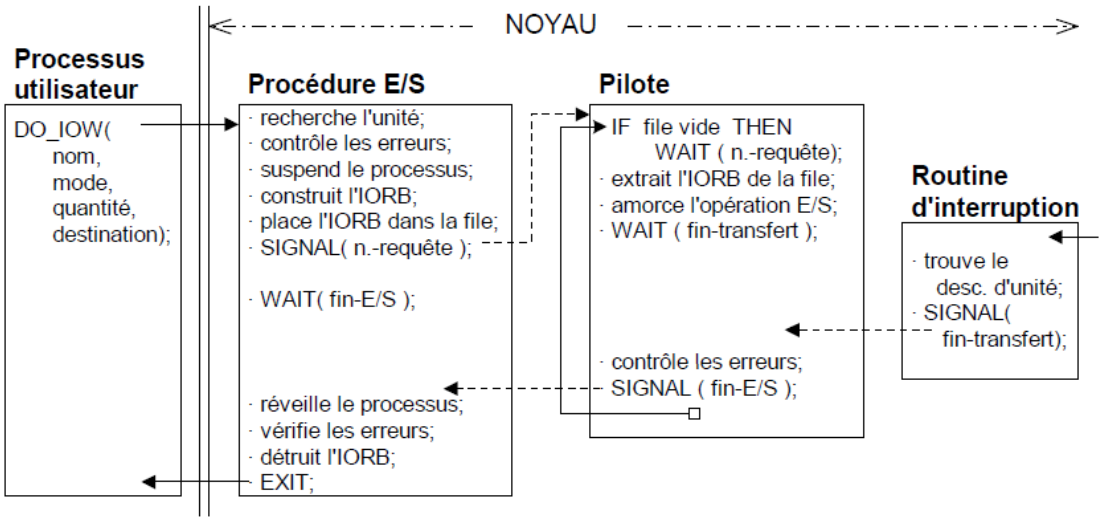
Ce cours discute des interfaces dans un système embarqué lorsqu'il y un système d'exploitation.

1 Généralités

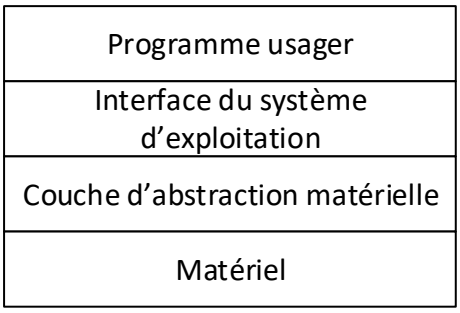
Dans un SMI avec système d'exploitation, le système d'exploitation fournit habituellement toutes les fonctions d'accès aux périphériques :

- Tout le code principal pour accéder aux périphériques est programmé par les gens qui programment le système d'exploitation.
- Toutes les interruptions reliées aux périphériques sont programmées par les gens qui programment le système d'exploitation.
- Tout le code relié aux périphériques est rattaché au système d'exploitation et il est toujours exécuté en mode superviseur ou super-utilisateur avec toutes les permissions possibles.
    - Un corollaire : toutes les interruptions sont exécutées en mode superviseur.

La figure suivante résume les accès aux périphériques dans un SMI avec système d'exploitation :



Toutes les sections qui suivent réfèrent à cette figure. Une autre façon de le voir est de voir que le système d'exploitation fournit une couche d'abstraction pour l'accès aux périphériques : le programmeur n'y a jamais accès directement.



2 Accès aux périphériques par les processus utilisateur

Le système d'exploitation offre des fonctions aux différents processus/programmeurs pour accéder aux périphériques. Ces fonctions sont souvent des fonctions similaires à celles d'accès aux fichiers. Par exemple, sous Linux, on retrouve File Open, File Write ou File Read pour accéder à un port série.

Lorsque le programmeur utilise un système d'exploitation, il doit utiliser un compilateur qui reconnaît ces fonctions, souvent uniformisées. Le programmeur doit connaître les prototypes de ces fonctions pour accéder aux périphériques.

Comme le compilateur ne sait pas nécessairement les adresses des fonctions d'accès aux périphériques, on utilise généralement des interruptions logicielles ou appel de service afin d'appeler les routines du système d'exploitation.

Lorsque le programmeur veut accéder aux périphériques et qu'il utilise un système d'exploitation préemptif, il doit faire attention à la préemption.

3 La procédure d'Entrée-Sortie

La procédure d'entrée/sortie vérifie d'abord si le périphérique existe et si l'accès est valide. Par exemple, il faudra ouvrir un port série (déterminer la vitesse de communication, réserver de la mémoire pour les entrées/sorties, réserver un numéro de référence (handle) pour le périphérique et plus) avant d'envoyer un message...

Ensuite, la procédure d'Entrée-Sortie suspend le processus. Cela fait référence à l'état des processus vu précédemment : un système d'exploitation efficace n'exécutera pas inutilement des processus qui attendent après différents périphériques...

Après avoir suspendu le processus, la procédure d'entrée sortie lance l'accès au périphérique en faisant appel aux fonctions du pilote de périphérique. Cet appel se fait souvent à l'aide de requêtes placées dans des files d'attentes…

Enfin, la procédure d'entrée sortie se met en attente du résultat de l'accès au périphérique. Cette attente n'est pas une attente active : la fonction est ré-exécutée seulement lorsque l'accès au périphérique est complété. À ce moment, la procédure d'E/S retourne le résultat de l'opération au processus usager.

4 Le pilote de périphérique

Un pilote de périphérique est constitué d'un ensemble de routines. Ces routines sont appelées par le système d'exploitation lors de certains évènements (pour traiter un IRP par exemple). On dit que les routines sont « callback », parce qu'elles sont appelées par le noyau sous certaines circonstances, pas par le pilote de périphérique.

Le pilote de périphérique doit s'initialiser et se retirer de la mémoire proprement. Par exemple, il doit faire le lien entre ses routines et les circonstances décrites plus bas lors de son initialisation.

Un pilote de périphérique doit gérer l'ajout ou le retrait de périphérique.

Un pilote de périphérique doit traiter les requêtes faisant des accès au périphérique (Create, Read, Write, Close, Contrôle de I/O). Les routines pour de tels IRP sont appelées routines de dispatch car elles ne font souvent qu'un pré traitement comme de la vérification de paramètres avant de dispatcher l'exécution de la requête à une autre partie du pilote.

Un pilote de périphérique doit s'assurer que les requêtes arrivent une à une au périphérique.

Un pilote de périphérique doit gérer certains aspects du matériel comme les interruptions, le DMA et les appels de procédure différé.

La liste des fonctions d'un pilote pourrait s'allonger…

5 La routine d'interruption

La routine d'interruption contient du code intimement relié au contrôleur de périphérique et à ses registres. Souvent, le code dans l'interruption sera très court : on lèvera un drapeau, écrira un octet dans une structure de donnée ou on effectuera une tâche très brève…

Par rapport au système d'exploitation, il faut retenir les éléments suivants :

- Toutes les routines d'interruption pour les périphériques ont priorité sur l'interruption périodique du système d'exploitation décidant de la prochaine tâche à exécuter.
- La routine d'interruption est du code du système d'exploitation. Dans le cœur ARM, par exemple, on entre en mode "super-user" dès que l'on traite une interruption.
- Du code contenu dans la routine d'interruption doit mener au changement de l'état de la tâche ayant fait l'accès au périphérique.

6 Exemple

Voici un exemple détaillé illustrant tous les éléments précédents. Dans cet exemple, la tâche 1 veut transmettre une chaîne de caractère ayant 8 octets sur le port série et attendre que la transmission soit finie avant de continuer.

Code de la tâche 1	Procédure d'E/S

void Tache1(void){
 	
initTache1();
UART_Handle = FileOpen("COM1", &Params);
 	
While(1){
CodeDeTache1QuelconqueP1();
 	
//Accès au port série
GetMutex(&UARTMutex)
FileWrite(UART_Handle, Chaine, 8)
FreeMutex(&UARTMutex)
 	
CodeDeTache1QuelconqueP2();
}
	status FileWrite(int Handle, char* Data, int DataLen){
	
	if(Handle == HANDLE_NOT_VALID)
	  return BadHandleStatus;
	
	if(GetFilePermission(Handle, WritePermission) != WPermissionGranted)
	  return CannotWriteFileStatus;
	
	File[Handle].Driver.SendData(Data, DataLen);
	
	TacheActive->Peripherique = File[Handle].Driver.Name;
	TacheActive->Etat = EtatDeTache_AttendApresPeripherique;
	while(TacheActive->Etat == EtatDeTache_AttendApresPeripherique)
	{}
	
	return File[Handle].Driver->SendDataResult;
	}

Code (fonctions) du Pilote de périphérique	Interruption de UART en transmission

void SendData(char* Data, int DataLen){
int i;
 	
for(i = 0;. i < DataLen; i++)
{
  MetDansBufferCirculaire(Data[i]);
}
 	
if(REG_UART_TX_STATUS == TX_IDLE)
{
 REG_UART_TX = LisOctetDuBufferCirculaire();
}
}
 	
void OnTxDone()
{
int i;
 	
SendDataResult = FileWriteOK;
for(i = 0; i < OS_GetNombreDeTache(); i++)
{
 if(OS_Tache[i].Peripherique == UART)
 {
  OS_Tache[i].Etat = EtatDeTache_Pret;
 }
}
}
	void UART_TX_ISR(void)
	{
	int ProchainOctetATransmettre;
	
	ProchainOctetATransmettre = LisOctetDuBufferCirculaire();
	if(ProchainOctetATransmettre != PAS_DOCTET)
	{
	REG_UART_TX = ProchainOctetATransmettre;
	}
	else
	{
	  OnTxDone();
	}
	}

6.1 Remarques générales sur l'exemple

L'exemple n'est pas aussi étoffé que celui d'un système d'exploitation et beaucoup d'items ne sont jamais implémentés comme dans l'exemple. On utilise habituellement beaucoup plus de pointeurs et de ressources dans un OS classique. Toutefois, l'exemple de code illustre plus clairement les concepts vus précédemment.

Voici quelques items de l'exemple qui ne reflètent pas la réalité:

- Les paramètres des fonctions du système d'exploitation sont souvent plus élaborés. Par exemple, la fonction FileOpen qu'on retrouve au début reçoit souvent un nom de fichier, le répertoire du fichier, les modes d'accès au fichier (write, read) et autres.
- L'appel au code du système d'exploitation se fait habituellement avec une interruption logicielle (SVC) parce que l'adresse de la Procédure d'E/S n'est pas disponible pour le compilateur de l'application. Plutôt que d'appeler FileWrite(UART_Handle, Chaine, 8), le compilateur met les paramètres de l'appel dans des registres du microcontrôleur prédéterminés et utilise l'instruction SVC(#X) pour appeler la fonction FileWrite (#X est le numéro de la fonction)
- Le système de fichier est beaucoup plus clairement défini. L'exemple est flou sur les fonctions comme GetFilePermission.
- L'association entre le système de fichier et les pilotes de périphérique n'est jamais codée comme dans l'exemple.
- L'appel des fonctions implémentées dans le pilote de périphérique se fait souvent avec une file de requêtes et par des mécanismes mieux décrits.
- …

Bref, l'exemple sert à faire comprendre les points suivants :

- Le système d'exploitation implémente les fonctions d'accès aux périphériques. Ces fonctions sont disponibles pour les programmes de l'usager sous diverses formes, mais habituellement comme accès à des fichiers.
- L'usager voulant accéder à un périphérique doit idéalement gérer la ressource avec des mutexes
- Lorsqu'une tâche accède à un périphérique, cette tâche devient bloquée et elle n'est plus exécutée par le système d'exploitation tant que l'accès au périphérique n'est pas fini. La fin de l'accès est signalé par une interruption qui changera l'état de la tâche afin d'en permettre l'exécution de nouveau.
- Les pilotes de périphérique et les interruptions reliées appartiennent au domaine du système d'exploitation

6.2 Code de la tâche 1

- La fonction FileOpen retourne un HANDLE. Un HANDLE est un numéro de référence (souvent un int) sur le fichier. Il sert à identifier le fichier lors des accès subséquents.
- GetMutex et FreeMutex sont nécessaires si plusieurs tâches veulent transmettre des octets sur le port série ou utiliser la ressource.
- Le prototype de la fonction FileWrite est très représentatif du prototype typique rencontré.

6.3 Procédure d'E/S

- File[Handle].Driver.SendData appelle une fonction du pilote de périphérique. Dans l'exemple, on assume que SendData est un pointeur de fonction qui a été initialisé lors du FileOpen
- Le code TacheActive->Etat = EtatDeTache_AttendApresPeripherique; suivi de while(TacheActive->Etat == EtatDeTache_AttendApresPeripherique){} est exécuté jusqu'à ce qu'une interruption du OS survienne. À partir de ce moment, le OS n'exécutera plus la tâche qui est maintenant dans un état "bloqué" et le microprocesseur ne devrait pas attendre inutilement dans la boucle while très longtemps.
- Après l'interruption du périphérique, l'état de la tâche changera. Le OS réexécutera la tâche et sortira du while(TacheActive->Etat == EtatDeTache_AttendApresPeripherique){}

6.4 Pilote de Périphérique

- Les fonctions du pilote de périphérique sont appelées par le OS lorsqu'on accède au périphérique
- C'est dans ces fonctions que se fera habituellement les accès aux registres des périphériques (REG_UART_TX et REG_UART_TX_STATUS dans l'exemple) et la gestion de l'accès (le buffer circulaire dans le cas présent)
- Ces fonctions préparent souvent le déclenchement d'une interruption qui indiquera que l'accès est terminé.
- Après l'interruption, l'exécution des fonctions du pilote de périphérique a normalement lieu.

6.5 Interruption

- Il faut noter que la tâche effectuée dans l'interruption est courte…
- Il faut noter que l'interruption cause un changement à l'état de la tâche pour celle-ci soit ré-exécutée après l'accès au périphérique.