Report Phase 2

Authors

Abstract

Un'implementazione dell'inizializzazione del nucleo, dello scheduler e dell'exception handler, il quale si occupa di chiamate di sistema e interrupt, del progetto del sistema operativo Panda+, in particolare la fase 2, ovvero il Livello 3 del progetto.

Documentazione delle funzioni

Per le varie funzioni che sono state create esiste una documentazione scritta all'interno del codice. Essa può quindi essere letta direttamente dai file ".h"; nel caso si desiderasse invece vedere una pagina web contenente questi dettagli, basterà eseguire il comando "make open_docs" dopo aver installato doxygen.

Se si vogliono comprendere le funzioni nei minimi dettagli, anch'esse sono state commentate per chiarire il loro funzionamento interno. Questi commenti non verranno mostrati nella pagina web creata dal comando sopra, ma saranno solo visibili all'interno del codice, nei file ".c".

Scelte progettuali

Struttura delle directory

Abbiamo suddiviso la cartella della consegna nel seguente modo:

- `pandos/phase1`
  - Contiene tutti i file della fase 1 del progetto
- `pandos/phase2`
  - Contiene la fase 2 del progetto
- `pandos/stdlib`
  - Contiene alcune funzioni di libreria utili al debug e allo sviluppo del progetto

File contenuti in `phase2`

Abbiamo i seguenti file all'interno della cartella `pandos/phase2`:

- `main.c`
  - Contiene l'inizializzazione del kernel e l'entry point del nostro sistema operativo.
- `sheduler.c`
  - Contiene lo scheduler del nostro sistema operativo, oltre alle sue strutture dati e funzioni ausiliarie
- `exception.c`
  - Contiene la funzione che si occupa di gestire le eccezioni e varie funzioni ausiliarie
- `syscall.c`
  - Contiene le funzioni relative alla gestione delle systemcall
- `interrupt.c`
  - Contiene le funzioni relative alla gestione degli interrupt
- `devices.c`
  - Contiene i semafori dei devices e delle funzioni per poterci interagire

Oltre a questi file, all'interno di `pandos/phase2/include` vi sono i vari header di questi file, i quali hanno i commenti che formano la documentazione del progetto.

Scelte progettuali

Di seguito verranno elencate le scelte progettuali che abbiamo fatto per la fase 2 del progetto.

Inizializzazione del kernel

L'inizializzazione del kernel avviene nel file `main.c`, il quale contiene la funzione `main` che è l'entry point del nostro sistema operativo. Nella funzione `initKernel` vengono inizializzate le strutture dati del kernel, il passup vector e lo scheduler . Inoltre, viene creato il processo `test` e viene infine chiamato lo scheduler.

Scheduler

Lo scheduler utilizzato da questo sistema operativo è di tipo round-robin a singola priorità, con un timeslice di 5ms. Esso è implementato in `scheduler.c`. Ad ogni chiamata dello scheduler, il primo controllo viene fatto sulla coda ready: se essa non è vuota, viene preso il primo processo in coda e viene eseguito. Se invece la coda è vuota, vengono fatti dei controlli per decretare se il sistema ha concluso la sua esecuzione, se va messo in WAIT oppure se siamo in una situazione deadlock.

Gestione delle eccezioni

Le eccezioni gestibili dal kernel sono gli interrupt e le systemcall da 1 a 10. In ogni altro caso, viene eseguita una `PassUpOrDie` che termina il processo corrente nel caso non sia definita una struttura di supporto per quel processo, altrimenti procede con la gestione dell'eccezione utilizzando quest'ultima.

Gestione delle systemcall

Vengono gestite dal kernel le seguenti systemcall:

- `CREATEPROCESS`
- `TERMINATEPROCESS`
- `PASSEREN`
- `VERHOGEN`
- `DOIO`
- `GETTIME`
- `CLOCKWAIT`
- `GETSUPPORTPTR`
- `GETPROCESSID`
- `GETCHILDREN`

Per dettagli implementativi delle singole systemcall, si veda la documentazione delle funzioni.

Gestione degli interrupt

Dato che il nostro sistema operativo non gestisce interrupt annidati, abbiamo implementanto un `interruptHandler` che gestisce gli interrupt in modo sequenziale, in ordine di priorità. In questo modo, è possibile gestire un interrupt di priorità più bassa che è arrivato mentre si stava gestendo un interrupt di priorità più alta.

Per dettagli implementativi dell'handler, si veda la documentazione delle funzioni.

Gestione dei devices

Nel nostro sistema operativo sono presenti 40 device, ad ognuno dei quali è associato un semaforo. Per i terminali, abbiamo due semafori in quanto la scrittura e la lettura sono trattate come due operazioni distinte. Abbiamo inoltre un semaforo per l'interval timer, che viene utilizzato per la gestione del clock.

In totale sono quindi presenti 49 semafori.

Compilazione

Per compilare i file eseguire

```
make all
```

Per eseguire il sistema operativo, eseguire

```
make run
```

o

```
cd machine && umps3 Phase2
```

Per compilare in debug mode, eseguire

```
make debug
```