RELAZIONE PandOS+

ALEXANDRU NICOLESCU, SAMUELE BUSI

April 23, 2022

Initial

L'inizializzazione del kernel avviene in questa parte, seguendo precisamente le indicazioni del manuale, in particolare sono dichiarate le variabili globali utilizzate in tutto il programma: la gestione della priorità dei processi è affidata a due code separate.

Scheduler

Lo scheduler è implementato in modo minimale, i processi ad alta priorità non sono sottoposti a preemption e vengono immediatamente eseguiti, mentre quelli a bassa priorità vengono gestiti seguendo una politica round robin, la funzione "ContextSwitch()" è impiegata per chiarezza del codice; al momento dell'esecuzione viene segnato il TOD fornito dal timer per mantenere traccia del tempo che il processo passerà nella gestione di eccezioni.
I contatori mantenuti dal codice sono utilizzati per identificare lo stato attuale e attendere eventualmente i processi bloccati sui semafori dei dispositivi.

Exceptions

Exception handler che direziona la chiamata e salva lo stato Syscall handler controlla l'ingresso nella fase di chiamata di sistema, verificando di trovarsi in user mode, salvandosi i registri della rispettiva chiamata e invocando la funzione corretta.
Per quanto riguarda l'implementazione delle singole syscall, l'approccio scelto è stato di scrivere funzioni dedicate, cercando di utilizzare ottimamente il codice già scritto; ad esempio la funzione "block(int * semaphore_address)" impiegata per tutte le chiamate bloccanti; nella gestione delle operazioni di IO le informazioni relative al dispositivo sono ottenuti dall'indirizzo fisico fornito, grazie alla funzione "findDevice(memaddr adress)"; la terminazione di un processo con tutti gli eventuali figli è implementata con l'aiuto di "TerminateTree" che termina ricorsivamente tutti i processi figli invocando "TerminateSingleProcess".

Nel file è anche dichiarata “CopyPaste(state_t * copy; state_t * paste)” ovvero il metodo di nostra scelta per copiare i registri di stato dei processi alternativamente a una memcopy.

Interrupts

Gli interrupt sono amministrati in una funzione unica, che utilizzando il registro cause identifica il dispositivo e lo gestisce accordatamente: per le linee con più dispositivi attaccati viene invocata la funzione “whichDevice(unsigned int bitMap)” che utilizzando la bitMap identifica quello corretto, per tutti i dispositivi non terminali viene invocata la funzione “handleDeviceInterrupt()” che in questa fase del progetto è implementata in modo essenziale.
Gli interrupt sollevati dal Process Local Timer dopo aver effettuato l’acknowledgement, aggiornato stato e tempo trascorso verificano che il processo in esecuzione sia effettivamente a bassa priorità prima di gestire l’eccezione.
I terminali vengono gestiti con attenzione particolare per identificare se si tratti di un caso di scrittura o lettura (non implementata), i dispositivi di trasmissione sono stati da noi assegnati ai registri devreg[4][0-7] e quelli di ricezione ai successivi; l’operazione di V sul semaforo del terminale è re implementata per semplicità e per poter aggiornare il registro v0.

Conclusione

Questa fase è stata sviluppata seguendo rigorosamente le indicazioni fornite sul manuale, Cercando di scrivere codice chiaro e fedele alle specifiche, l’implementazione delle operazioni sui semafori di Verhogen e Proberen è presa dalle slides di concorrenza del corso e tutto il progetto tenta di allinearsi il più possibili alle nozioni teoriche apprese.