YAEOS

Yet Another Educational Operating System
Specifiche dell'esercitazione di Laboratorio
Sistemi Operativi
A.A. 2017-18

Phase2
v.0.3

Renzo Davoli

Phase2

- Lo scopo della phase 2 è di costruire il nucleo del kernel.
- Phase2 usa le strutture dati costruite in phase 1 e le funzioni fornite dalla ROM e dalla libreria di uARM
- Occorre implementare l’astrazione di processi sequenziali asincroni.

Phase 2

- Occorre implementare:
  - Lo scheduler (per fare in modo che i processi in esecuzione possano a turno usare il processore)
  - Una interfaccia di system call che consenta ai processi di chiedere operazioni al nucleo quali creazione/terminazione di processi, operazioni di sincronizzazione, input-output
  - Routine di gestione delle eccezioni per elaborare interrupt e trap, incluse operazioni per poter trasferire la gestione di alcune eccezioni ai livelli superiori dell'architettura del sistema operativo (passup)

Phase2: memoria e modo

- Il codice di phase2 verrà eseguito con la risoluzione degli indirizzi spenta (CP15_Control[0]=0) e in modo kernel.
- Questo significa che il programma userà gli indirizzi fisici in ogni accesso alla memoria e che tutta la memoria sarà accessibile.
- Il codice di phase2 deve però preservare la modalità di accesso alla memoria e il modo di esecuzione (kernel/user) dei processi.

Phase2: lo scheduler

- YAEOS supporta uno scheduler preemptive a priorità con aging per la CPU.
- Deve essere previsto anche un time-slice di 3 millisecondi.
- Ogni processo perde il controllo della CPU se rimane running per 3ms.
- Ogni 10ms la priorità dei processi nella coda ready viene incrementata di 1 (fino a raggiungere il valore di MAXPRIO).

Scheduler: controllo di deadlock

- Occorre implementare un semplice controllo di deadlock.
- Il sistema deve mantenere in una variabile il numero di processi attivi nel sistema e il numero di processi soft-blocked cioè in attesa del completamento di operazioni di Input-Output.
- Quando non ci sono processi nella ready queue possono darsi tre casi:
  - Non ci sono più processi attivi = SHUTDOWN
    - Terminare l'esecuzione con HALT
  - Ci sono processi soft-blocked
    - Mettete il sistema in WAIT STATE
  - Non ci sono processi soft-blocked = DEADLOCK
    - Terminare l'esecuzione con PANIC

Nucleo inizializzazione

- Il nucleo inizia l'esecuzione dalla funzione main, come ogni programma C.
  - Inizializzate le aree NEW per interrupt e trap nel frame riservato dalla ROM
    - PC = indirizzo routine di gestione
    - SP = RAMTOP
    - CPSR: tutti gli interrupt disabilitati
  - Inizializzate le strutture dati che avete costruito in phase1.
  - Inizializzate tutte le variabili del nucleo (es. I contatori dei processi)
  - Inizializzate I semafori che vi serviranno nel nucleo
  - Create il PCB per il primo processo (vedi prossimo lucido)
  - Chiamate lo scheduler

PCB del primo processo

- Allocate un PCB e riempite I campi come segue:
  - Interrupt abilitati
  - Priorità 0
  - Memoria virtuale spenta
  - Kernel mode
  - SP = RAMPTOP – FRAMESIZE (il penultimo frame della memoria)
  - PC: l'indirizzo della funzione test (che sarà l'entry-point del codice di phase2test).
    - extern void test();
    - … = (memaddr) test;

Stato iniziale

- Il nucleo
  - Viene caricato a partire dall'indirizzo 0x00008000 (seconda pagina di RAM, la prima è la pagina riservata per il codice della ROM di sistema)
  - PC = 0x00008000
  - SP = RAMTOP
    - Ma se leggete lo SP sarà di qualche byte minore perché c'è il record di attivazione del main stesso

Ora inizia il gioco...

- La chiamata dello scheduler costituisce la fine della fase di inizializzazione
- Il controllo non tornerà più al main.
- Il codice del nucleo verrà eseguito come effetto di trap o di interrupt.

Trap di tipo system-call/breakpoint

- Quando avviene una trap di questo tipo viene attivata la routine di gestione che avrete caricato nell'area NEW relativa.
- System-call si riconoscono dalle trap di tipo breakpoint per un diverso valore del codice della eccezione (exception code).
- Alcune system call verranno gestite dal nucleo.
- Le rimanenti system call e le eccezioni di tipo breakpoint dovranno essere inoltrate al relativo gestore del livello superiore (se è stato stabilito) oppure causano la teminazione del processo.

System call del nucleo

- SYS1: Create Process
- SYS2: Terminate Process
- SYS3: P
- SYS4: V
- SYS5: Specify Trap Handler
- SYS6: Get Times
- SYS7: Wait for clock
- SYS8: IO operation
- SYS9: GetPIDS
- SYS10: WaitChild

SYS1: Create Process

- Int SYSCALL (CREATEPROCESS, state t *statep, int priority, void **cpid)

  Questa system call crea un nuovo processo come figlio del chiamante. Il program counter, lo stack pointer, e i flag di configurazione (modo, vm etc) sono indicati nello stato iniziale. Se la system call ha successo il valore di ritorno è zero altrimenti è -1.

  Se la chiamata ha successo cpid contiene l’identificatore del processo figlio (indirizzo del PCB).

SYS2: Terminate Process

- int SYSCALL (TERMINATEPROCESS, void * pid)
- Questa system call termina il processo indicato (o il processo chiamante se pid==NULL) e tutta la progenie del processo indicato.
- Ritorna 0 se l'operazione ha avuto successo, -1 in caso contrario (ovviamente se pid==NULL oppure è il pid proprio o di un proprio avo, la chiamata se ha successo NON ritorna).

SYS3: P e SYS4: V

- void SYSCALL (SEMP, int *semaddr)
- void SYSCALL (SEMV, int *semaddr)

  Queste chiamate realizzano le operazioni P e V su un semaforo.
- Il valore del semaforo è memorizzato nella variabile di tipo intero passata per indirizzo
- L’indirizzo della variabile agisce da indentificatore del semaforo.

SYS5: Specify Trap Handler

- int SYSCALL (SPECHDL, int type, state_t *old, state_t *new)
- Questa chiamata registra quale handler di livello superiore debba essere attivato in caso di trap di Syscall/breakpoint (type=0), TLB (type=1) o Program trap (type=2).
- Il significato dei parametri old e new è lo stesso delle aree old e new gestitre dal codice della ROM: quando avviene una trap da passare al gestore lo stato del processo che ha causato la trap viene posto nell'area old e viene caricato o stato presente nell'area new.
- La system call deve essere richiamata una sola volta per tipo.
- Se la system call ha successo restituisce 0 altrimenti -1.

SYS6: Get Times

- void SYSCALL (GETTIME, cputime_t *user, cputime_t *kernel, cputime_t *wallclock)
- Questa system call restituisce il valore di tre “tempi” del processo:
  - Il tempo usato dal processo in modalità user
  - Il tempo usato dal processo in modalità kernel (gestione system call e interrupt relativi al processo)
  - Il tempo trascorso dalla prima attivazione del processo.

SYS7: Wait for clock

- void SYSCALL (WAITCLOCK)
- Questa system call sospende il processo fino al prossimo tick di 100ms.
- Lo pseudoclock produce un tick ogni 100ms (esatti) e risveglia tutti i processi che hanno chiesto la wait for clock.
- Occorre fare in modo che non si accumulino gli errori di sincronizzazione dello pseudo clock (la scadenza del prossimo tick deve sempre essere posta a 100ms esatti dal precedente).

SYS8: IO operation

- unsigned int SYSCALL (IODEVOP, unsigned int command, unsigned int *comm_device_register)
- Attiva l'operazione di I/O copiando il comando (command) nel campo comando del device register (*comm_device_register).
- Il chiamante verrà sospeso fino a completamento della operazione di input output, il valore di ritorno è il valore della registro di stato status (che indica quindi il successo o meno dell'operazione).
- Notate che i terminali sono device "doppi" c'è un campo command per ricevere e uno per trasmettere.
- (un solo processo alla volta accede ad uno specifico device, i processi si sincronizzano tramite sezioni critiche)

SYS9: GetPIDS

- Void SYSCALL(GETPIDS, void **pid, void **ppid)
- Questa system call restituisce il pid del processo stesso e del processo genitore.
- Se il campo pid o ppid è NULL il valore corrispondente non viene restituito.
- Per il processo radice *ppid è NULL.

SYS10: WaitChild

- Void SYSCALL(WAITCHLD)
- Questa system call aspetta la terminazione di un processo figlio.

SYS1-SYS10 in user mode

- Le system call dal numero 1 al numero 10 sono riservate a processi in kernel mode.
- Se vengono chiamate da un processo in user mode il processo chiamante deve fare passup di una trap di tipo reserved instruction se è stato definito un gestore per questo tipo di trap, altrimenti terminare

Breakpoint e System call > SYS10

- Devono essere inoltrati al gestore di livello superiore se presente, altrimenti causano la terminazione del processo (come se il processo facesse una SYS2 con parametro NULL).

Trap di tipo TLB e Program Trap

- Devono essere inoltrati al gestore di livello superiore corrispondente se presente, altrimenti causano la terminazione del processo (come se il processo facesse una SYS2 con parametro NULL).
- In caso di riattivazione a seguito della gestione di un Trap (e.g. Page Fault), l'istruzione che ha causato il trap va ripetuta.

Gestione degli Interrupt

- Occorre gestire le linee da 2 (interval timer) a 7 (terminali).
- Gli interrupt con numero più basso hanno priorità più alta.
- Utilizzate un semaforo per ogni device per “risvegliare” il processo che ha richiesto l’operazione di I-O con la SYS8 (due semafori per i terminali che sono device “doppi”).
- Notate che le linee di interrupt sono relative a tutti i device dello stesso tipo, occorre leggere il valore del CP15_Cause.IP per vedere quale effettivamente abbia causato l’interrupt.

Identificazione dei Processi

- I process ID sono indirizzi, in particolare si usa l'indirizzo del PCB come identificativo del processo.
- Vengono usati da SYS1, SYS2 e SYS9.

Pseudo Clock

- Deve produrre un tick ogni 100ms.
- Non si può “contare” i time slice perché i processi possono non completarne l’uso quando si bloccano per I/O o per sincronizzarsi con altri processi.
- Occorre sempre caricare l’interval timer con il tempo minimo fra la fine del time slice e il tempo mancante al prossimo tick dello pseudo clock.
- L’elaborazione dell’interrupt impiega anch’essa tempo quindi:
  - Può capitare che all’arrivo di un interrupt dell’interval timer sia al tempo stesso sia terminato il time slice sia la deadline per il tick dello pseudoclock
  - Occorre mettere la prossima deadline a 100 ms dalla precedente (non dal momento del caricamento) per evitare l’accumulazione di errori.

Stati del Processore

- Quando avviene un interrupt o un trap lo stato del processo corrente viene salvata nell'area OLD relativa al tipo di evento accaduto e viene caricato lo stato presente nell'area NEW.
- Ogni informazione relativa al processo che ha causato la trap o che era in esecuzione al momento dell'interrupt si trova nell'area OLD.
- Se il processo deve essere sospeso occorre copiare lo stato presente nell'area old nell'apposito campo del PCB del processo.
- Facendo LDST dello stato del processo, questo viene riattivato.

Consegna

- Ci sono tre deadline di consegna:
  - Domenica 10 giugno 2018 ore 23.59
  - Domenica 29 luglio 2018 ore 23.59
  - Domenica 23 settembre 2018 ore 23.59
- CONSEGNARE IL PROPRIO PROGETTO (un unico file .tar.gz) NELLA DIRECTORY DI CONSEGNA ASSOCIATA AL PROPRIO GRUPPO:
- **`/home/students/LABSO/2018/submit_phase2.june/lso2018az…`**
- **`/home/students/LABSO/2018/submit_phase2.july/lso2018az…`**
- **`/home/students/LABSO/2018/submit_phase2.final/lso2018az…`**
- CONSEGNARE ENTRO LA DEADLINE FISSATA.
- VERIFICARE CHE L'ARCHIVIO .TAR.GZ SIA COMPLETO

Consegna

- Cosa consegnare:
  - Sorgenti del progetto (TUTTI)
  - Makefile per la compilazione (eventuali file di AutoMake/Cmake)
  - README con istruzioni di compilazione
  - Documentazione (scelte progettuali)
  - File AUTHORS

- Occorre inserire commenti nel codice per favorire la leggibilità e la correzione ...
- PROGETTI non COMMENTATI NON SARANNO VALUTATI.
- PROGETTI che contengono troppi commenti inutili verranno valutati negativamente
- VERRÀ valutato il grado di professionalità nella gestione del codice