BiKayaOS

Progetto Sistemi operativi – Università di Bologna, 2020

# Intruduzione

BiKaya è un sistema operativo basato su un’architettura a 6 livelli proposta da Edsger Dijkstra e ideata per l’istruzione alla programmazione di sistemi operativi.

Il progetto compiuto consiste nell’implementazione in due architetture (uARM e uMPS) del Kernel (livello 3) dell’architettura Kaya, completo di supporto per 8 syscall e gestori di interrupt hardware, trap ed eccezioni TLB.

# Struttura

I file sono divisi in modo funzionale, in base al contenuto:

* *asl* – funzioni per operazioni sui semafori/liste di semafori;
* *exceptions* – gestori e funzioni di aiuto per le eccezioni (syscall, trap e TLB);
* *init* – funzioni di startup e inizializzazione;
* *interrupts* – gestore e funzioni di aiuto per gli interrupt;
* *pcb* – funzioni per operazioni sui blocchi di controllo dei processi (pcb) /liste di pcb;
* *utils* – funzioni varie usate in più contesti.

Tutte le syscall e le funzioni che astraggono gli handler degli interrupt dei device seguono la struttura:

int call(state\_t\* callerState)

per essere compatibile con il critical\_wrapper, la funzione che funge da wrapper e aggiorna il tempo kernel di tutti i processi.

# Descrizione funzoni

## ASL

* semd\_t\* getSemd(int \*key):

ritorna il semaforo in ASL con chiava pari a key, null se non trovato

* void initASL():

inizializza la tabella dei semafori dei processi e li inserisce della lista dei semafori liberi, ed initializza la chiave dei semafori dei dispositivi

* void insertSem(semd\_t\* semaphore):

inserisce semaphore nella ASL, con priorita in base alla chiave

* int insertBlocked(int \*key,pcb\_t\* p):

inserisce p nel semaforo in ASL con chiave key, oppure se il semaforo non è in ASL ne alloca uno e lo inserisce in ASL

* pcb\_t\* removeBlocked(int \*key):

rimuove e ritorna il primo pcb bloccato dal semaforo con chive key. Se il semaforo diventa vuoto lo rimuove da ASL e lo si reinserisce in nella lista dei semafori liberi

* pcb\_t\* removeBlockedonDevice(int\* key):

rimuove il processo bloccato nel semaforo del device con chiave pari a key. Se il semaforo diventa vuoto lo rimuove da ASL

* pcb\_t\* outBlocked(pcb\_t \*p):

ritorna e rimuove il pcb puntato da p dal suo semaforo, se il semaforo diventa vuoto viene tolto da ASL e inserito nella lista dei semafori liberi.

* pcb\_t\* DevicesOutBlocked(pcb\_t\* p):

ritorna e rimuove il pcb puntato da p dal suo semaforo per I device, se il semaforo diventa vuoto viene tolto da ASL

* unsigned int isBlocked(pcb\_t\* p):

ritorna true se p è bloccato in un semaforo

* pcb\_t\* headBlocked(int \*key):

ritorna il primo pcb bloccato nel semaforo con chiave key, senza rimuoverlo

* void outChildBlocked(pcb\_t \*p):

rimuove p e tutti I pcb radicati in p dai rispettivi semafori

## Exceptions

### Handlers:

* void trapHandler():

Gestore trap interrupts, evocato quando una trap è sollevata.

* void syscallHandler():

Gestore syscall, evocato quando è chiamata alla funzione SYSCALL. Legge la cause di chiamata e chiama la syscall richiesta.

* void TLBManager():

Gestore TLB, evocato quando c’è una violazione delle aree di memoria.

### Syscalls:

* int get\_cpu\_time(state\_t\* callerState):

SYS1. Ritorna tempo user, kernel e totale del processo chiamante.

* int create\_process(state\_t\* callerState):

SYS2. Crea un nuovo processo figlio del chiamante.

* int kill(state\_t\* callerState):

SYS3. Termina il processo richiesto e l’intera progenie.

* int verhogen(state\_t\* callerState):

SYS4. Effettua una V sul semaforo passato.

* int passeren(state\_t\* callerState):

SYS5. Effettua una P sul semaforo passato.

* int do\_IO(state\_t\* callerState):

SYS6. Richiede un’operazione I/O sul registro indicato.

* int spec\_passup(state\_t\* callerState):

SYS7. Specifica e salva un nuovo handler per syscall/trap/TLB.

* int get\_pid\_ppid(state\_t\* callerState):

SYS8. Ritorna il pid e/o il pid del parent del processo attuale.

### Funzioni di supporto:

* void recursive\_kill(pcb\_t\* process):

supporto per la (kill) SYS3. Termina I processi figli di process.

## Init

* void newArea(unsigned int address, unsigned int handler):

Imposta il registro di stato puntato da address per gestire syscall/interrupts con l’handler puntato da handler.

* void newProcess(memaddr functionAddr, int priority):

Imposta l’area di memoria per un nuovo processo da far partire dopo lo startup.

* void initROM():

Imposta gli handler per gestire syscall, trap, violazioni TLB e interrupts.

* void initData():

Istanzia le strutture dati per processi e semafori.

* void initialProcess():

Crea tutti i processi da far partire.

* int main():

Chiama tutte le funzioni precedenti e infine lo scheduler.

## Interrupts

### Handler:

* void interruptHandler():

Gestore interrupt principale, evocato quando un interrupt è sollevato. Controlla le richieste in entrata e le inoltra al corretto subhandler mantenendo la priorità del device.

Sub-handlers:

* int timerHandler(state\_t\* oldstatus):

Gestore per interrupt del interval timer.

* Handler di device (diskHandler, tapeHandler, netHandler, printerHandler, termHandler):

Inoltrano la richiesta a devHandler passando la corretta linea di interrupt.

### Funzioni di supporto:

* void devHandler(int line, state\_t\* oldstatus):

Gestore per interrupt di linea>3, gestisce sia gli interrupt generici che quelli di terminale.

* void exitInterrupt(state\_t\* oldstatus):

Operazioni di chiusura del gestore dell’interrupt. Copia il vecchio stato nello stato del processo attuale e reinserisce il processo corrente nella coda dei processi.

## PCB

* void initPcbs(void):

Inizializza la tabella dei pcb e li inserisce nella lista dei pcb liberi

* void freePcb(pcb\_t \*p):

aggiunge il pcb puntato da p nella lista libera

* pcb\_t \*allocPcb(void):

estrae dalla lista dei pcb liberi un pcb e lo iniziallizza, ritorna il pcb inizializzato.

* void mkEmptyProcQ(struct list\_head \*head):

inizializza una listhead, con entrambi I campi di head che puntano a se stessa

* int emptyProcQ(struct list\_head \*head):

ritorna true, se la lista puntata da head è vuota

* void insertProcQ(struct list\_head \*head, pcb\_t \*p):

inserisce il pcb puntato da p nella lista puntata da head, a seconda della priorita di p

* pcb\_t \*headProcQ(struct list\_head \*head):

ritorna il primo pcb nella lista puntata da head

* pcb\_t \*removeProcQ(struct list\_head \*head):

rimuove e ritorna il primo pcb dalla lista puntata da head

* pcb\_t \*outProcQ(struct list\_head \*head, pcb\_t \*p):

rimuove e ritorna il pcb puntato da p nella lista puntata da head, ritorna null se p non appartiene alla lista

* int emptyChild(pcb\_t \*this):

ritorna true se il pcb puntato da this non ha figli

* void insertChild(pcb\_t \*prnt, pcb\_t \*p):

inserisce p come figlio di prnt

* pcb\_t \*removeChild(pcb\_t \*p):

rimuove e ritorna il primo figlio di p

* pcb\_t \*outChild(pcb\_t \*p):

ritorna e rimuove p dalla lista dei figli di p->parent

## Utils

* void mymemcpy(void \*dest, void \*src, int n):

Funziona come una memcpy, copia l’area da memoria da dest a src.

* int critical\_wrapper(int (\*call)(), state\_t\* callerState, cpu\_time start\_time, pcb\_t\* currentProcess):

Tiene traccia del tempo user e kernel in un modo ordinato. Chiama la funzione passata tramite il puntatore call e ritorna TRUE o FALSE a seconda se deve essere chiamato lo scheduler o ripresa l’esecuzione del processo corrente alla fine della zona critica.

* cpu\_time update\_user\_time(pcb\_t\* currentProcess):

Funzione di aiuto del critical\_wrapper. Aggiorna il tempo utente e ritorna il TOD di chiamata.

* void set\_return(state\_t\* caller, int status):

Imposta il registro di ritorno di una syscall del caller a status.

* void set\_command(devreg\_t\* reg, unsigned int command, int subdevice):

Imposta il comando command nel registro di device in reg. Il subdevice rappresenta se il terminale è in trasmissione o in ricezione.

* void get\_line\_dev(devreg\_t\* reg, int\* line, int\* dev):

Ritorna linea e numero di device dato il registro di un device. Il calcolo è fatto trovando l’offset tra l’indirizzo del registro e l’inizio dell’area di memoria dei registri di device, applicando il dividendo per la dimensione di ogni blocco di registri dei device (<dimensione registro> \* <numero di linee ogni interrupt>) trovare la linea e applicando il modulo per trovare il numero di device.

* unsigned int get\_status(devreg\_t\* reg, int subdevice);

unsigned int tx\_status(termreg\_t \*tp);

unsigned int rx\_status(termreg\_t \*tp):

Ritorna lo stato di un device generico o terminale (tx\_status per il transm status, rx\_status per il recv status).

* void debug(int, int):

Funzione di debug per leggere parametri con coppia chiave/valore.

* semd\_t \* getSemDev(int line, int devnum, int read):

Ritorna la struttura dati del semaforo device richiesto da line, devnum e read.

* void verhogenKill(pcb\_t\* process):

Effettua le operazioni necessarie sui semafori a seguito di una SYS3.

* void verhogenDevice(int line, unsigned int status, int deviceNumber, int read):

Effettua una V sul semaforo del device, a seguito della gestione di un interrupt.

* void ACKDevice(unsigned int\* commandRegister):

Invia un segnale di ACK al device specificato (scrive il valore di ACK nel registro specificato).

* int pid\_in\_readyQ(pcb\_t\* pid):

Ritorna true se il pcb pid è nella coda dei processi in attesa.

* int set\_register(unsigned int reg, unsigned int val):

Imposta nel registro reg il valore val.

* int log2(unsigned int n):

Implementazione ricorsiva del logaritmo binario. Supporto della funzione getDeviceNumber.

* int lowest\_set(int number):

Trova il LSB di number, necessario come supporto di getDeviceNumber.

* int getDeviceNumber(int line):

Ritorna il numero di device del device alla linea che sta sollevando l’interrupt line. Per ottenere la linea di interrupt partendo dalla causa si prende la posizione ordinale da destra (trovata con log2) del bit di minimo valore (trovata con lowest\_set) della causa di interrupt.