# Documentazione

Nikolas Acquaviva, Stefano Carminati, Matteo Manuelli, Michele Monteferrante

Aprile 2022

## 1 Introduzione

GitHub Repository: https://github.com/NikolasAcquaviva/OSProject2122

Questa documentazione serve a spiegare come abbiamo deciso di affrontare la seconda fase del progetto di Sistemi Operativi.

In questa fase abbiamo dovuto costurire lo scheduler, l'exeptionhandler, l'interrupthandler e il main del nostro sistema operativo: vediamo ora nel dettaglio le nostre implementazioni.

## 2 Init.c

Init.c inizializza il Bios: parte con l'inizializzare il passupVector con i pc e sp adeguati nei loro registri, successivamente crea e setta l'inizializzazione dei semafori e dei processi e infine chiama lo scheduler. Dopo questa chiamata Init.c esaurisce la sua funzione e passa totalmente il controllo.

## 3 Scheduler.c

Lo scheduler si occupa di gestire i processi: più precisamente di decidere qual è il prossimo processo da eseguire, controllando se l'ultimo processo era ad alta priorità e se ha rilasciato le risorse con yield(), poiché bisogna evitare che tali processi riprendano immediatamente dopo l'operazione yield(), e di definire in quale stato si troverà la macchina fra HALT, WAIT e, in caso di deadlock PANIC.

## 4 Exeptionhandler.c

### 4.1 GeneralExeptionHandler

Ottiene il contenuto del registro cause, per poi avere il codice eccezione, in base al suo valore scopriamo che tipo di eccezione dobbiamo gestire:

Se è 0 si tratta di un interrupt, minore di 3 un TLB, uguale a 8 è una Syscall e se minore di 12 è una Trap.

## 4.2 PassUpOrDie

Si occupa di tutte le eccezioni che non siano Syscall e Interrupts, tramite index scopriamo se siamo nel caso di un'eccezione generale o in una page fault, dopodiché se il processo corrente non ha struttura di supporto, lo terminiamo, altrimenti settiamo i giusti parametri della stessa.

## 4.3 SyscallExeptionHandler

Gestisce le syscall, controllando prima il codice della syscall, se è positivo, ma valido, si esegue la PassUpOrDie, altrimenti se il codice è nel range negativo con user mode oppure non valido simuliamo un program trap, infine se il codice è negativo e nel range (-1, -10) facciamo uno switch con le varie syscall alla fine del quale dobbiamo caricare lo stato salvato nella bios data page e incrementare di una word il program counter per non avere il loop infinito di syscalls.

Analizziamo ora le varie Syscall:

### 4.3.1 CreateProcess

Crea il processo tramite la funzione "allocPcb()", gli assegna i campi che la funzione non considera, essendo una funzione della prima fase, successivamente lo inserisce come figlio del processo corrente, nella coda ad alta o bassa priorità in base al suo campo prio e infine ne ritorna il pid.

### 4.3.2 TerminateProcess

Abbiamo, per pulizia, ma anche per necessità nelle Syscall successive, deciso di creare quattro funzioni ausiliarie che in questa Syscall fanno la maggior parte del lavoro; Terminate controlla il primo valore, ovvero il pid passato in input, se è 0 termina il processo chiamante, altrimenti tramite FindProcess trova il processo attraverso il pid e lo termina.

Analizziamo meglio come avviene la terminazione del processo:

se il processo da terminare non ha figli, viene chiamata la funzione Die:

che prende in input il processo e un intero: se l'intero passato è 1 e il processo non ha padre, si esegue un semplice outchild, così da rimuovere direttamente la radice, altrimenti si controlla se il campo semAdd è vuoto, se non lo è si guardano i deviceSemaphore e se si trova il processo si decrementa il numero di processi bloccati da essi, altrimenti si controllano i semafori binari e se il processo non è bloccato si incremente il campo semAdd, altrimenti si usa removeBlocked e si rimuove dalle code di priorità, infine, se si è entrati nel caso semAdd != NULL si usa outBlocked. Se invece il campo semAdd risulta NULL allora si controlla se il processo corrente è quello da terminare: se non è così si rimuove dalla coda dei processi pronti. Infine si libera il processo tramite FreePcb, si decrementa il numero di processi e si pone il processo corrente uguale a NULL, se il numero di processi raggiunge lo 0.

Se ha figli invece si chiama la recursiveDie, che tramite visita in profondità per ogni pcb appartenente al sottoalbero richiama la Die citata precedentemente.

#### 4.3.3 Passeren

Inizialmente controlla se facendo InsertBlocked la lista dei processi liberi sia vuota, se lo è va in PANIC, altrimenti esegue un outBlocked per sbloccare il processo appena bloccato. Dopodiché se semaddr è 0 allora blocchiamo il processo e, se si tratta di un deviceSemahore, incrementiamo il softBlockCount e chiamiamo lo scheduler; altrimenti se il puntatore alla lista dei processi con chiave semAddr è diverso da NULL, si rimuove il processo dalla coda dei bloccati al semaforo e si inserisce nelle liste di priorità, ovviamente decrementando il numero di softBlockCount se è un deviceSemaphore; altrimenti si decrementa semAddr.

#### 4.3.4 Verhogen

Questa Syscall è equivalente alla Passeren, ma esegue i processi opposti, mantenendo gli stessi controlli. Infatti se la Passeren esegue una P operation, Verhogen esegue una V operation.

#### 4.3.5 Do - Io

Controlla se il device è il terminale, che ha registri di trasmissione e ricezione, o di tipo generale, ovvero con solo command register, se è il terminale setta la line a 4, poiché è costante, se è di altro tipo incrementa la line con un ciclo for annidato per ogni device, dopodiché si setta a cmdValue il comando del device. Tramite la variabile isRecvTerm controlla se il device era di ricezione e in caso setta l'indice a line*8 + numDevice + 8, altrimenti a line*8 + numDevice. Dopodiché inserisce il processo nella coda dei processi bloccati associata al semaforo ed incrementa il softBlockCount, decrementando il valore dei deviceSemaphore, richiama lo scheduler e, dopo la chiamata, si controlla quale device è e si ritorna lo stato, altrimenti, se isRecvTerm è 1, si ritorna lo stato di ricezione del terminale, altrimenti si ritorna lo stato di trasmissione.

#### 4.3.6 GetCPUTime

Ritorna il tempo accumulato dal processore in microsecondi usato per il processo richiesto tramite questa semplice formula: currentProcess-¿ptime += currentTime - startTime.

#### 4.3.7 WaitForClock

Performa una Passeren su un interval-timer semaphore.

#### 4.3.8 GetSupportData

Ritorna la struttura di supporto del processo corrente, se non c'è ritorna NULL.

#### 4.3.9 GetProcessID

Se il parent del processo corrente è 0 ritorna il pid del processo corrente, altrimenti quello del parent.

#### 4.3.10 Yield

Modifica lo stato del processo corrente e lo inserisce nella coda di priorità corretta, dopodiché setta una variabile che userà lo scheduler al processo corrente e richiama lo scheduler stesso.

## 5 InterruptHandler.c

Entriamo in InterruptExceptionHandler che tramite getInterruptInt ottiene la linea richiesta dall'interrupt e ne controlla il valore:

se è 0 va in PANIC;

se è 1 fa un update dello stato del current process, per poi inserirlo nella coda di priorità corretta, richiamando poi lo scheduler;

se è 2 sblocca tutti i processi bloccati dall'interval-timer semaphore, decrementa il softBlockCount adeguatamente e inserisce il processo nella coda di priorità corretta.

se invece è maggiore di 2 passiamo il controllo alla NonTimerHandler, un'altra funzione:

questa esegue un ulteriore controllo iniziale:

se la linea è compresa fra 2 e 7 si invia un ACK e salva lo stato da ritornare;

se la linea è 7 si casta il device register al terminal register, se il terminale non è pronto a ricevere salva lo stato da ritornare, invia un ACK e setta come pronta la ricezione di un interrupt, altrimenti salvo solo lo stato da ritornare e invio un ACK.

Se si è entrati nella NonTimerHandler, dopo i controlli sulle linee si eseguono le seguenti operazioni:

troviamo l'indirizzo del deviceSemaphore, lo incrementiamo, e lo sblocchiamo;

se c'era almeno un processo bloccato inserisce lo stato da ritornare nel registro v0, sblocca il processo, ricalcola il tempo del processo, diminuisce il softBlockCount e, se il processo è diverso da quello corrente, lo inserisce nella coda di priorità corretta poi se il processo corrente è uguale a NULL chiama lo scheduler, altrimenti se il processo sbloccato ha una priorità maggiore rispetto al processo corrente, copia lo stato del processore nel pcb del processo corrente e lo inserisce nella coda di priorità adeguata, dopodiché chiama comunque lo scheduler, altrimenti carica il vecchio stato;

infine se il processo corrente è uguale a NULL richiama lo scheduler, altrimenti carica lo stato del BIOS.