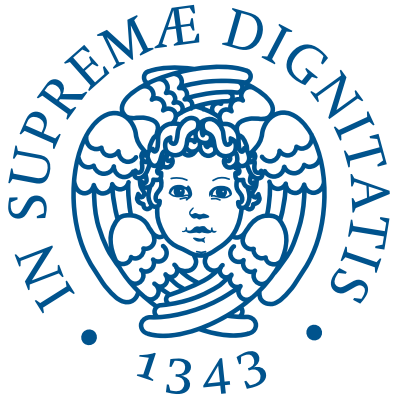

UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Informatica

Implementazione del meccanismo dello swapping di processi nel nucleo didattico

Relatore:

Prof: Giuseppe Lettieri

Candidato:

Luca Di Gasparro

ANNO ACCADEMICO 2024/2025

Indice

Capitolo 1

Il meccanismo dello swapping di processi

1.1 La storia

Con l'aumento della complessità e delle dimensioni delle applicazioni utente e l'introduzione di sistemi multiprogrammati e multiutente, è storicamente sorto il problema della gestione della memoria primaria di un sistema: essa è di dimensioni limitate, ma deve poter essere utilizzata per l'esecuzione di un numero arbitrario di processi utente e sistema. Questa problematica assunse particolare rilevanza nei sistemi operativi degli anni '70 e '80, quale Unix: l'idea per un meccanismo risolutivo riguardò l'utilizzo di un dispositivo di memoria secondaria, quale un hard disk. Trasferendo su di esso la memoria di un processo (operazione di swap-out), altri processi possono sfruttare la memoria primaria liberatasi; il sistema ha poi il compito di riportare in memoria primaria tali processi da essa rimossi (operazione di swap-in). Tale meccanismo, interagendo con periferiche esterne i cui tempi di elaborazione sono significativamente superiori rispetto a quelli di una unità di elaborazione centrale (CPU), richiede inoltre la scelta di politiche che riducano al minimo il numero di interazioni con la memoria secondaria.

Nei sistemi operativi moderni [2], questa soluzione ha lasciato il posto a tecniche più avanzate di paginazione su domanda, le quali possiedono una granularità maggiore sulla quantità di memoria da trasferire in memoria secondaria: questo approccio riduce l'overhead di I/O, consente l'esecuzione di processi la cui memoria non è completamente disponibile in memoria primaria e si adatta dinamicamente alla richiesta di memoria da parte dei processi. Tuttavia, può ancora essere utilizzato in alcuni casi di uso particolarmente intensivo della memoria primaria, insieme ad altre tecniche quali l'Out-Of-Memory-Killer.

1.2 Un esempio di implementazione: Unix

Unix, uno dei primi sistemi ad implementare tale meccanismo, ha definito delle politiche per garantirne un funzionamento efficace e corretto [1].

Uno swap-out può venire richiesto in tre differenti casi:

- Viene invocata una `fork()`, primitiva per la creazione di un nuovo processo, ma la memoria primaria è insufficiente
- Viene invocata una `brk()`, primitiva per l'estensione del segmento dei dati di un processo, ma la memoria primaria è insufficiente
- Il sistema decide di effettuare uno swap-in, ma la memoria primaria è insufficiente

La politica per la scelta del processo vittima dell'operazione di swap-out (cioè quello che subirà il trasferimento in memoria secondaria) è a base prioritaria: il nucleo implementa una funzione la quale, dato un processo, ne restituisce una precedenza in base alla priorità medesima del processo, alla sua niceness e alla quantità di tempo in cui è rimasto caricato in memoria primaria; verrà scelto poi il processo con tale priorità calcolata maggiore. Vengono quindi esaminati tutti i processi del sistema, preferendo quelli attualmente bloccati (ad esempio, su una coda semaforica) e poi quelli in coda pronti.

Per quanto riguarda lo swap-in, ne possono essere soggetti solamente processi che hanno subito uno swap-out e che sono attualmente pronti per andare in esecuzione. Tra tutti i processi in memoria secondaria, viene scelto il processo che da più tempo ha subito lo swap-out.

Il processo swapper (ovvero quello che si occupa di eseguire le operazioni di swap-in e swap-out) è quindi strutturato come un ciclo infinito in cui ricerca un processo di cui fare swap-in, secondo la politica descritta. In caso di ricerca fallimentare, la sua esecuzione viene sospesa, per essere ripresa quando la ricerca produrrà in maniera certa un risultato; altrimenti, tenta di eseguire lo swap-in di tale processo, eventualmente preceduto da uno swap-out nel caso di memoria primaria insufficiente.

Capitolo 2

Ambiente di sviluppo

2.1 Il nucleo didattico

Il nucleo utilizzato per l'implementazione è un nucleo a 64 bit utilizzato per finalità didattiche nel corso di Calcolatori Elettronici, sviluppato e mantenuto dal prof. Giuseppe Lettieri [3].

2.1.1 I moduli

Il nucleo è composto di tre moduli, programmi a sé stanti e non collegati con gli altri:

- **sistema**: contiene la realizzazione dei processi, inclusa la gestione della memoria fisica e virtuale
- **I/O**: contiene le routine di ingresso/uscita che permettono di utilizzare le periferiche collegate al sistema (tra cui l'hard disk utilizzato per lo swapping)
- **utente**: contiene il singolo programma che l'utente può scrivere e che verrà poi eseguito (in questo programma è comunque possibile creare un numero arbitrario di processi, che saranno poi eseguiti concorrentemente sull'unica CPU disponibile per il sistema)

Modulo sistema ed I/O vengono eseguiti col processore a livello sistema, in un contesto privilegiato, mentre il modulo utente a livello utente.

Inoltre, il modulo sistema è definito atomico: durante la sua esecuzione le interruzioni esterne mascherabili sono disabilitate, e non implementa pertanto un meccanismo di accesso concorrente alle strutture dati da esso mantenute. Ciò

implica che un processo non può tipicamente sospendere la propria esecuzione all'interno del modulo sistema per poi riprenderla in seguito al suo interno, poiché potrebbe lasciare le strutture dati in uno stato inconsistente.

2.1.2 Schedulazione e code di processi

La schedulazione del nucleo si basa su una priorità statica assegnata ad ogni processo al momento della sua creazione. Viene realizzata una coda `pronti`, in cui sono inseriti tutti i processi attualmente pronti per andare in esecuzione. Altre code di processi, rilevanti per lo sviluppo del lavoro presente, sono quelle semaforiche, in cui un processo permane durante il suo stato di blocco in seguito alla chiamata, ad esempio, di una primitiva di `sem_wait`.

2.1.3 La memoria virtuale

La memoria virtuale di ogni processo è suddivisa in varie sezioni [4]:

- **sistema/condivisa**: contiene la finestra sulla memoria fisica e le traduzioni per gli indirizzi corrispondenti a registri di periferiche.
- **sistema/privata**: contiene la pila sistema del processo.
- **IO/condivisa**: contiene il modulo I/O (sezioni *.text*, *.data* e heap I/O).
- **utente/condivisa**: contiene il modulo utente (sezioni *.text*, *.data* e heap utente).
- **utente/privata**: contiene la pila utente del processo.

2.1.4 Shutdown del sistema

Lo shutdown del sistema è realizzato mediante il processo `dummy`: questo controlla ciclicamente che la variabile `processi` contenga un valore strettamente positivo; in caso contrario effettua lo shutdown. Tale variabile viene incrementata ad ogni invocazione della primitiva `activate_p()`, normalmente utilizzata nel modulo utente per la creazione di nuovi processi.

2.2 Strategia ed obiettivi implementativi

L'idea implementativa alla base del lavoro svolto richiama quella di Unix: uno swap-out viene richiesto nel caso di memoria primaria insufficiente per completare la creazione di un nuovo processo o una operazione di swap-in; uno swap-in viene richiesto, invece, quando la memoria primaria libera è sufficiente

per riportare in memoria un processo precedentemente trasferito in memoria secondaria.

Di conseguenza, lo swap-out, se richiesto nel primo caso precedente, deve essere una operazione bloccante per il processo padre affinchè completi la creazione del processo figlio. Si vuole quindi implementare un blocco nel modulo sistema, rilassando la condizione di atomicità precedentemente presentata: in fase implementativa, sarà pertanto necessario verificare il mantenimento di uno stato consistente, qualunque sia l'ordine di schedulazione o di blocco dei processi al suo interno.

Data la struttura modulare del nucleo, la gestione delle richieste delle operazioni di swapping deve essere svolta nel modulo sistema, mentre l'interazione con la memoria secondaria avviene mediante un processo swapper operante nel modulo I/O. Ciò rende necessaria la comunicazione tra i due moduli: il processo swapper deve ricevere dal modulo sistema dei descrittori delle operazioni di swapping da eseguire, mentre il modulo sistema dovrà gestire il blocco del processo swapper.

Schematicamente, quindi, il processo swapper richiede al sistema un descrittore per una operazione da eseguire; effettua i trasferimenti I/O necessari e comunica al modulo sistema il termine dell'operazione associata al particolare descrittore. Tutte le operazioni riguardanti la realizzazione dei processi, la memoria virtuale e la gestione delle code sono eseguite dal sistema prima e dopo le operazioni di I/O.

Capitolo 3

Implementazione nel nucleo didattico

Questo capitolo ha l'obiettivo di spiegare i passaggi logici svolti durante la fase implementativa nel nucleo didattico per realizzare gli obiettivi presentati in precedenza. Vengono quindi esposti i problemi riscontrati, le soluzioni adottate ed eventuali pregi e difetti emersi da una loro analisi critica. L'analisi effettuata rimane comunque ad alto livello, senza spiegazione dettagliata del codice medesimo. Questo è rinvenibile in una repository online [6], in cui è stato caricato il sorgente del nucleo didattico nella versione utilizzata per lo sviluppo. Le aggiunte inserite per il lavoro corrente sono racchiuse in dei commenti identificati dal tag *TESI*. Sono presenti inoltre commenti nel codice sorgente qualora il suo funzionamento fosse poco chiaro.

3.1 Memoria virtuale

La memoria di un processo soggetta a swapping comprende le sue sezioni private, quindi la pila di sistema e la pila utente, le cui dimensioni sono predefinite e fisse.

Per quanto concerne le tabelle delle pagine, è necessario gestire due bit dei descrittori delle pagine: il bit di presenza ed il bit di scrittura [5].

Il bit di presenza indica la validità della traduzione dell'indirizzo virtuale associato a una determinata pagina, permettendo l'intercettazione di page-fault su pagine non mappate in memoria fisica: non viene disattivato durante un'operazione di swap-out poiché un processo che lo subisce non può essere eseguito prima del corrispondente swap-in, evitando a priori i page-fault.

Il bit di scrittura, invece, viene attivato dalla Memory Management Unit (MMU)

in risposta a un'operazione di scrittura sulla pagina: viene resettato al momento dello swap-out in quanto le modifiche sulla pagina sono state trasferite in memoria secondaria. Tale approccio consente un'ottimizzazione volta alla riduzione del numero di trasferimenti tra memoria primaria e secondaria: se un processo che ha già subito uno swap-out in precedenza viene selezionato come vittima per un ulteriore swap-out, verranno trasferiti in memoria secondaria esclusivamente quei frame su cui è stata effettuata una scrittura tra le due operazioni, poiché per gli altri la copia più recente è già stata salvata.

Inoltre, i descrittori delle pagine contengono anche un campo con l'indirizzo fisico del frame corrispondente a tale pagina: al momento dello swap-in vengono sovrascritti con quelli dei nuovi frame allocati per la memoria privata del processo. Sono inoltre gli indirizzi che dovranno essere comunicati al processo swapper, il quale, avendo una memoria virtuale differente da quella del processo del quale svolge l'operazione di swapping, avrà traduzioni differenti degli stessi indirizzi virtuali; dovrà quindi comunicare alla periferica esterna di memoria secondaria direttamente gli indirizzi fisici della memoria primaria su cui effettuare le operazioni di ingresso/uscita.

3.2 Hard disk

Come memoria secondaria, viene utilizzato l'hard disk di sistema. Esso è disponibile esclusivamente come memoria di swap; quindi, né l'utente né il sistema (per scopi differenti dallo swapping) possono accedervi. La memoria di massa viene suddivisa in un numero predefinito di partizioni fisse contigue, ognuna di dimensione sufficiente a contenere la memoria soggetta a swapping di un processo (di dimensione fissa come indicato precedentemente). Tali partizioni sono rappresentate nel modulo sistema mediante una bitmask di dimensione appropriata, che ne monitora lo stato (libere o occupate dalla memoria di un processo). Inoltre, ad un processo che subisce swap-out viene assegnata una partizione, che non verrà liberata fin quando il processo medesimo non termina.

I trasferimenti tra memoria primaria e secondaria vengono effettuati tramite Direct Memory Access (DMA) per evitare il consumo di cicli di CPU non necessari e consentire ad altri processi di essere eseguiti durante tali trasferimenti.

3.3 Politiche

Per quanto riguarda lo swap-in, nello schedulatore di sistema si controlla se il numero di frame attualmente liberi in memoria primaria è maggiore o uguale a quelli occupati dalla memoria soggetta a swap di un processo, moltiplicati per una costante parametrica. In caso positivo, si esegue lo swap-in del processo a maggiore priorità tra quelli non caricati in memoria primaria; altrimenti, si

richiede uno swap-out per liberare memoria per tale processo, rimandando lo swap-in alla prossima evenienza.

Tale costante parametrica è da scegliersi in base al tipo di programmi utente da eseguire. Ridurla implica incrementare il numero di operazioni effettuate e ridurre la latenza del sistema (definita come il tempo medio che un processo spende in attesa dello swap-in), il che è desiderabile se si hanno processi fortemente legati ad operazioni di I/O, poichè si sospenderebbero in attesa della loro terminazione e, divenendo bloccati, sarebbero le vittime ideali per ulteriori swap-out, o se il numero di processi presenti nel sistema è ridotto (un numero elevato di processi causerebbe code di operazioni di swapping molto lunghe, determinando thrashing e starvation). Incrementarla indica invece una diminuzione delle operazioni di swapping ma una maggiore latenza, ideale nei casi speculari a quelli descritti e tendenzialmente preferibile. Inoltre, se statisticamente la durata dei processi utente è breve, è consigliabile mantenere una costante elevata, dato che la memoria primaria viene liberata naturalmente dalla terminazione dei processi medesimi.

Dai test effettuati, in cui vi è un'elevata frequenza nella creazione di processi, una costante di valore due è risultata il trade-off ottimale.

Per quanto riguarda lo swap-out, la politica adottata è quella di ricercare un processo utente bloccato su un semaforo e, in mancanza di questi, cercarne uno tra quelli pronti. Si itera difatti sui semafori utente e, trovato un semaforo con una coda di processi sospesi non vuota, si estrae il processo a minore priorità di tale coda. Si sceglie di non effettuare swapping di processi di sistema: in essi rientrano driver e processi esterni le cui operazioni, in quanto risposte a interruzioni esterne, sono time-sensitive, e per cui uno swap-in può costituire un ritardo eccessivo. Inoltre, in tale categoria, potrebbero rientrare anche processi periodici e demoni, o che in generale devono compiere azioni immediate o con tempistiche prefissate. Di conseguenza, si ritiene una buona politica il lasciarli sempre in memoria primaria. Si sceglie, inoltre, di non considerare come possibili vittime i processi bloccati nella coda del timer: di questi, difatti, è sicuro il risveglio in un intervallo di tempo prestabilito, oltre il quale saranno in coda pronti. Sarebbe ideale realizzare un'euristica che valuti l'intervallo di attesa minimo di un processo nella coda del timer tale per cui possa essere considerato una possibile vittima, considerando la lunghezza della attuale coda delle operazioni di swapping e il tempo medio di completamento di tali operazioni.

Una possibile criticà di tale politiche risiede nella seguente casistica: si vuole effettuare lo swap-in di un processo, ma la memoria primaria risulta insufficiente, richiedendo un'operazione di swap-out. Se le code semaforiche dovessero risultare vuote, il processo vittima viene scelto dalla coda pronti. Se la vittima possiede una priorità maggiore del processo oggetto di swap-in, per mantenere la coerenza col meccanismo prioritario, tale swap-out non dovrebbe essere consentito. Non si effettuano tali controlli per evitare la starvation della coda dei processi in attesa di swap-in.

L'idea alla base di queste politiche è di evitare l'eccessivo riempirsi della coda dei processi in attesa di swap-in, poichè, se la memoria primaria è soggetta a grande richiesta, la latenza aumenta, così come il numero di operazioni di swap-out richieste per riportare in memoria tali processi, generando un meccanismo di reazione positiva che porterebbe velocemente al thrashing; si preferisce quindi decretare il fallimento di uno swap-out per mancanza di processi vittima.

3.4 Sospensione nello swap-out

Dall'analisi delle strategie implementative, è emerso che uno swap-out può essere richiesto in due situazioni differenti.

Durante la creazione di un processo, è necessario che il processo padre sia sospeso in attesa della terminazione dell'operazione di swap-out, affinché continui con la creazione del processo figlio e ritorni poi nel modulo utente.

Se richiesto per liberare memoria per uno swap-in, tuttavia, non è appropriato sospendere il processo che ha invocato l'operazione di swap-out, poiché non esiste una relazione diretta con il processo vittima dello swap-out (lo swap-in, si ricorda, viene invocato nello schedulatore di sistema indifferentemente dal processo attualmente in esecuzione). L'adozione di uno swap-out bloccante comporterebbe anche la perdita dell'atomicità dello schedulatore di sistema e di conseguenza di tutte le primitive che contengono una chiamata allo schedulatore medesimo. Poiché il nucleo del sistema operativo non è progettato per gestire situazioni di non atomicità, ciò potrebbe portare a inconsistenze nelle code di processi e, in generale, nello stato del sistema. A conferma di ciò, si pensi ad un processo utente che invoca una `delay()`: verrebbe in questo modo inserito nella coda del timer. Al termine della primitiva, viene ovviamente effettuata una chiamata allo schedulatore di sistema. Se si effettua inizialmente il controllo sullo swap-in e tale swap-in richiedesse uno swap-out, il processo si bloccherebbe: sarebbe quindi presente in due code differenti. Se invece tale controllo venisse effettuato in seguito al cambio del processo in esecuzione, non si potrebbe realizzare il blocco in quanto il processo scelto per l'esecuzione utilizzerà ancora il contesto e la pila sistema del processo che è stato appena rimosso dall'esecuzione: il cambio di contesto effettivo viene difatti eseguito nella routine di `carica_stato` all'uscita dal modulo sistema.

Pertanto, è necessario implementare uno swap-out non bloccante, la cui unica differenza con quello bloccante risiede nel fatto che non ci sarà nessuna sospensione del processo che lo invoca.

3.5 Stato di un processo durante le operazioni di swapping

Nel modulo sistema è implementata una ulteriore coda di processi, `to_swap_in`, destinata a contenere tutti i processi pronti per andare in esecuzione ma non caricati in memoria primaria; viene quindi tenuta separata dalla coda `pronti`, in cui sono presenti invece i processi pronti ad andare in esecuzione e già caricati in memoria primaria.

Quando un processo viene scelto come vittima di swap-out, ci possono essere due casi:

- Scelto da `pronti`: viene rimosso da tale coda ed inserito in `to_swap_in`
- Scelto da una coda semaforica: non può venire rimosso, altrimenti non potrebbe essere risvegliato mediante una `sem_signal()`. Se risvegliato mentre non è caricato in memoria primaria o durante l'operazione di swap-out, si redirige l'inserimento da `pronti` a `to_swap_in`

Da ciò emerge che se un processo vittima è stato scelto da una coda semaforica, ma non viene risvegliato, allora non può essere soggetto a swap-in: tale comportamento è voluto poiché si ritiene non ottimale riportare in memoria un processo ancora bloccato, precludendo l'utilizzo della memoria primaria a processi che invece potrebbero già andare in esecuzione.

Un processo attraversa vari stati nel mentre, identificati nel descrittore di processo dai seguenti flag:

- `swapping_out`: settato da quando viene scelto come vittima di swap-out a quando tale operazione non è terminata
- `in_ram`: settato quando la memoria del processo è caricata in memoria primaria

Il flag `swapping_out` viene utilizzato quando si ricerca un processo di cui eseguire lo swap-in: se la vittima proviene da `pronti`, viene inserita immediatamente in `to_swap_in`, ma logicamente tale processo non può subire ancora l'operazione di swap-in poiché è ancora oggetto della precedente.

Entrambi i flag vengono utilizzati nella scelta della vittima di swap-out: nel caso di vittima scelta da una coda semaforica, e non ancora risvegliata dalla sospensione, non può essere considerato nuovamente come vittima di un altro swap-out. Vengono inoltre utilizzati per intercettare l'inserimento in `pronti` di processi che stanno attualmente subendo swap-out o che lo hanno già terminato, redirigendolo quindi in `to_swap_in` (esempio di una `sem_signal` che risveglia un processo non in memoria primaria).

Infine, quando la vittima subirà uno swap-in, verrà nuovamente inserita in `pronti`.

3.6 Strutture dati

Il sistema adotta diverse strutture dati per monitorare le operazioni di swapping richieste e per comunicarle al processo swapper. Le richieste vengono mantenute in una coda ordinata a priorità decrescente, implementata mediante una lista: nel caso di uno swap-out, la priorità corrisponde a quella del processo che richiede lo swap-out, mentre nel caso di uno swap-in essa è determinata dalla priorità del processo che deve subire lo swap-in. Nel caso di swap-out non bloccante richiesto per liberare memoria in vista di uno swap-in, la priorità è quella del processo di cui si vuole eseguire lo swap-in: come evidenziato in precedenza, difatti, non c'è correlazione tra esso e quello che richiede lo swap-out. Di conseguenza, quando il processo swapper richiede la prossima operazione da svolgere, verrà estratta la richiesta con la priorità più alta, senza distinzione tra swap-in e swap-out.

Per garantire una chiara distinzione delle responsabilità tra il modulo di sistema e il modulo I/O, anche a livello di informazioni scambiate, sono stati implementati due descrittori per le operazioni di swapping: uno di interfaccia, comunicato al modulo I/O, contenente solo le informazioni necessarie per l'esecuzione dell'operazione, ed uno di controllo, gestito invece dal sistema, in cui vengono memorizzati il corrispondente descrittore di interfaccia, la priorità della richiesta e i dati necessari per l'implementazione della coda. A tale coda vengono anche associati un contatore per tenere traccia del numero di elementi in essa presenti ed una coda di processi, per implementare il meccanismo di blocco del processo swapper nel caso di richiesta di una nuova operazione da svolgere con la coda delle richieste vuota, e del suo conseguente risveglio al primo inserimento in tale coda. Di conseguenza, tale coda di processi conterrà al massimo un solo processo, ovvero lo swapper. Il descrittore di interfaccia dovrà contenere anche il PID dei processi oggetto delle operazioni di swapping: quando il processo swapper termina un'operazione, lo notifica al modulo sistema mediante il descrittore di interfaccia identificativo; il modulo sistema, per eseguire le operazioni terminali, necessita del PID del processo oggetto dell'operazione.

Il descrittore di interfaccia include un array contenente gli indirizzi fisici coinvolti nell'operazione di swapping per i motivi già esposti. È possibile fare uso di un array in quanto la dimensione della memoria di un processo soggetta a swapping è di dimensione predefinita, come indicato precedentemente. Inoltre, il descrittore contiene il primo indice di blocco (secondo il Logical Block Addressing) dell'hard disk da cui deve essere effettuata l'operazione richiesta, corrispondente al primo LBA della partizione dedicata dal sistema al processo oggetto dell'operazione corrente.

Per quanto riguarda la memorizzazione dei descrittori, i descrittori di interfaccia sono inclusi staticamente nel descrittore di processo, mentre i descrittori di controllo sono allocati dinamicamente sullo heap di sistema quando necessario. Questa scelta consente di ridurre l'overhead associato all'uso dello heap durante le operazioni di swapping, poiché i descrittori di interfaccia sono già presenti nel descrittore del processo coinvolto nell'operazione, considerando anche che i processi subiscono tali operazioni a coppia, nel senso prima di swap-out, poi sicuramente anche di swap-in.

Ridurre l'uso dello heap è coerente con l'ambiente operativo del nucleo, che dispone di risorse limitate, e contribuisce a minimizzare il rischio di fallimenti delle operazioni di swap-in e swap-out a causa della mancanza di memoria dinamica. D'altra parte, allocare i descrittori di controllo sullo heap offre una maggiore flessibilità al sistema, consentendo di richiedere risorse aggiuntive solo in caso di utilizzo intensivo della memoria primaria, quando il numero di operazioni di swap aumenta.

Una possibile strategia per ridurre ulteriormente l'overhead associato alle operazioni di allocazione e deallocazione dei descrittori di controllo potrebbe consistere nel mantenere tali strutture dati in una coda, piuttosto che deallocarle immediatamente. In questo modo, si potrebbe verificare la presenza di descrittori liberi nella coda prima di effettuare nuove allocazioni, ottimizzando l'uso della memoria. Questo approccio non solo migliorerebbe l'efficienza, ma contribuirebbe anche a mantenere un utilizzo più sostenibile delle risorse di sistema.

3.7 Processo swapper

Il processo `swapper` ha la medesima struttura di un processo esterno: è costituito da un ciclo infinito, in cui richiede al modulo sistema un descrittore di interfaccia di una operazione di swapping e la completa, comunicando poi al sistema la corrispettiva terminazione. Sarà il sistema a gestire il blocco dello `swapper` nel caso in cui la coda delle operazioni di swapping sia vuota, e il suo successivo risveglio.

Un aspetto critico relativo al processo `swapper` è la sua creazione. Nel nucleo sono disponibili due primitive per la creazione di processi: `activate_p`, che crea un processo generico e lo include nel conteggio totale dei processi attivi nel sistema, influenzando così il comportamento di shutdown del sistema; e `activate_pe`, che crea un processo esterno associandolo a una specifica richiesta di interruzione, escludendolo dal conteggio dei processi attivi.

Per il processo `swapper`, si desidera un comportamento simile a quello fornito da `activate_p`, poiché non è associato a richieste di interruzione esterne. Tuttavia, essendo un processo demone, non deve essere presente nel conteggio dei processi. A tal fine, è stata introdotta la primitiva `activate_daemon`, che combina le

funzionalità desiderate di entrambe le primitive precedenti.

Capitolo 4

Conclusioni

Da questa implementazione del meccanismo dello swapping, è emerso come esso sia una tecnica comunque efficace nel gestire situazioni particolarmente critiche di gestione della memoria primaria. Tuttavia risulta poco flessibile ed introduce una latenza molto elevata nel sistema: se la dimensione della memoria di un processo soggetta a swapping è composta di un numero consistente di frame, i tempi di attesa dovuti alle operazioni in DMA diventerebbero insostenibili, comportando un ritardo maggiore anche nella gestione della coda delle operazioni di swapping nel modulo sistema. Motivo per cui, come detto inizialmente, tecniche con granularità di trasferimento verso la memoria secondaria maggiore, quali la paginazione su domanda, si dimostrano più efficaci.

Bibliografia

[1] Maurice J. Bach. 1986. The design of the UNIX operating system. Prentice-Hall, Inc., USA.

[2] Andrew S. Tanenbaum and Herbert Bos. 2014. Modern Operating Systems (4th. ed.). Prentice Hall Press, USA.

[3] Giuseppe Lettieri. Introduzione al sistema multiprogrammato `https://calcolatori.iet.unipi.it/resources/nucleo.pdf` Accessed: 2025/05/02

[4] Giuseppe Lettieri. Implementazione della memoria virtuale `https://calcolatori.iet.unipi.it/resources/paginazione-implementazione.pdf` Accessed: 2025/05/02

[5] Giuseppe Lettieri. Paginazione `https://calcolatori.iet.unipi.it/resources/paginazione.pdf` Accessed: 2025/05/02

[6] Luca Di Gasparro. Codice sorgente `https://github.com/sensatore/nucleo_swapping`