YAEOS - Gruppo 1 - Phase 1

Documentazione e scelte progettuali

Componenti del gruppo e funzioni implementate:
-Federico De Giorgio : Allocazione e deallocazione dei PCB;
-Valerio Leuzzi : Gestione delle code dei PCB;
-Giuseppe Montalbano : Gestione dell'abero dei PCB;
-Francesco Ballestrazzi : Gestione Active Semaphore Hash Table (ASHT);

ALLOCAZIONE E DEALLOCAZIONE DEI PCB

- **initPcbs**():
Inizializza la pcbFree in modo da contenere tutti gli elementi della pcbFree_table.
In initPcbs viene solamente inizializzato un intero i a zero che diventerà poi il parametro della funzione CreaLista insieme all'indirizzo di pcbfree_h.

- **freePcb**(pcb_t *p):
Inserisce il PCB puntato da p nella lista dei PCB liberi (pcbFree).
Per l'implementazione di questa funzione ho scelto di inserire l'elemento puntato da p in testa alla lista e riaggiornare la testa in modo da tornare ad essere il primo elemento.

- pcb_t *__allocPcb__():
Restituisce NULL se la pcbFree è vuota.
Altrimenti rimuove un elemento dalla pcbFree, inizializza tutti i campi (NULL/0) e restituisce l'elemento rimosso.
Come nella funzione precedente ho scelto di rimuovere l'elemento in testa e di riaggiornare la pcbfree_h, inoltre per inizializzare il pcb che viene ritornato dalla funzione uso una funzione esterna initPcbNULL(pcb_t *p) che, come da specifiche, pone a zero o NULL tutti i campi del pcb puntato. Ifine ritorno il pcb rimosso.

Per inizializzare la lista dei pcb liberi è stata utilizzata una funzione esterna void CreaLista che prende come parametri un doppio puntatore ed un intero.
L'intero viene inizializzato a zero nella funzione principale initPcbs e serve per scorrere l'array di pcb e non andare oltre la lunghezza massima MAXPROC, mentre il doppio puntatore è l'indirizzo di memoria della testa della lista. La funzione fa si che il valore puntato da head (ciè il puntatore pcbfree_h nella prima scansione ricorsiva) prenda il valore dell'indirizzo di memoria dell'elemento i-esim dell'array; viene poi richiamata la funzione con parametri incrementati i+1 e &((*head)->p_next) fino a che i diventa uguale al numero massimo di processi.

GESTIONE DELL'ALBERO DEI PCB

Oltre alla gestione dei PCB attraverso code, la struttura può essere gestita attraverso alberi n-ari di PCB. Tre elementi definiscono un nodo di albero: p_first_child, p_sibling, p_parent.
p_first_child definisce il primo figlio delle lista di tutti i figli del PCB, accessibili e uniti da p_sibling. Ogni processo figlio ha un puntatore al padre (p_parent).

La gestione dell'albero dei PCB avviene attraverso tre funzioni principali più altre due funzioni ausiliarie:

- void **insertSib**(pcb_t *p, pcb_t *sib)
Inserisce p come fratello piu' a destra della lista dei fratelli puntata da sib. Questa è una funzione ausiliaria utilizzata nelle funzioni principali.

- struct pcb_t ***removeSib(**pcb_t *p, pcb_t *sib)
Rimuove il PCB puntato da p dalla lista dei fratelli puntata da sib. Restituisce il PCB puntato da p se questo fa parte della lista dei fratelli altrimenti NULL. Quindi scorre la lista dei fratelli verificando se p appartiene alla lista. Questa è una funzione ausiliaria e servirà per le funzioni principali.

- void **insertChild**(pcb_t *parent, pcb_t *p)
Inserisce il PCB puntato da p come figlio del PCB puntato da parent. Se parent ha dei figli p verrà inserito come ultimo figlio, il fratello più a destra (vedi **insertSib**(pcb_t *p, pcb_t *sib)).

- struct pcb_t ***removeChild(**pcb_t *p)
Rimuove il primo figlio del PCB puntato da p. Se p non ha figli, restituisce NULL.

- struct pcb_t ***outChild**(pcb_t *p)
Rimuove il PCB puntato da p dalla lista dei figli del padre. Se il PCB puntato da p non ha un padre, restituisce NULL. Altrimenti restituisce l'elemento rimosso (cioé p). A differenza della removeChild(), p può trovarsi in una posizione arbitraria (ossia non è necessariamente il primo figlio del padre). Nel caso in cui p è il primo figlio del padre allora la funzione richiama la removeChild(). Invece se è uno dei fratelli la funzione richiama removeSib().

GESTIONE DELLE CODE DEI PCB

- void **insertProcQ**(pcb_t **head, pcb *p):
inserisce l'elemento puntato da p nella coda dei processi puntata da head. L'inserimento avviene tenendo conto della priorità di ciascun pcb in modo che alla fine la coda dei PCB abbia un ordine decrescente.

- pcb_t ***headProcQ**(pcb_t *head):
restituisce l'elemento in testa alla coda dei processi puntata da head. Ritorna NULL se la coda non ha elementi.

- pcb_t ***removeProcQ**(pcb_t **head):
rimuove il primo elemento dalla coda dei processi puntata da head. Ritorna NULL se la coda è vuota. Altrimenti ritorna il puntatore all'elemento rimosso dalla lista.

- pcb_t ***outProcQ**(pcb_t **head, pcb_t *p):
rimuove il PCB puntato da p dalla coda dei processi puntata da head. Se p non è presente nella coda restituisce NULL.

- void **forallProcQ**(pcb_t *head, void fun(pcb_t *pcb, void *), void *arg):
richiama la funzione fun per ogni elemento della lista puntata da head.

Le funzioni relative alla gestione delle code dei PCB sono state implementate esclusivamente in modo ricorsivo, come indicato nella specifica del progetto, e senza l'utilizzo di funzioni ausiliarie.

ASHT

- int **hashFun**(int *key):
Ritorna un intero che corrisponde all'indice di posizione della chiave key nell'ASHT.

- semd_t * **cercaInListaDiTrabocco**(semd_t *head, int *key):
Cerca il semd_t con identificativo key nella lista puntata da head.
Ritorna il puntatore a tale semd_t se viene trovato nella lista.
Ritorna NULL altrimenti.

- void **insertInListaDiTrabocco**(semd_t *sem, int *i):
Inserisce sem in testa alla lista di indice i nella semdhash.

- void **initASL_recursive**(int *i):
Inizializza la lista dei semd_t liberi scorrendo la semd_table.
L'intero i serve a scorrere l'intera semd_table senza uscire dalla sua dimensione MAXSEMD.

- void **insertInSemdFree**(semd_t *sem):
Inserisce il semd_t puntato da sem nella lista puntata da semdFree_h (inserimento in testa).

- semd_t ***removeFromSemdFree**():
Rimuove e ritorna la testa della lista dei semd liberi.
Se la lista è vuota ritorna NULL.

- semd_t ***removeDaTrabocco**(semd_t **head, semd_t *sem):
Rimuove il semd_t puntato da sem dalla lista con testa head.
Ritorna NULL se la testa non punta a nessuna lista o se sem non è presente nella lista.
Ritorna sem se è stato trovato e rimosso dalla lista.

- int **insertBlocked**(int *key, pcb_t *p):
Viene inserito il PCB puntato da p nella coda dei processi bloccati associata al semaforo con chiave key.
Se il semaforo corrispondente non è presente nella ASHT, alloca un nuovo SEMD dalla lista di quelli liberi e lo inserisce nella ASHT, settando i campi in maniera opportuna. Se non è possibile allocare un nuovo SEMD perché la lista di quelli liberi e' vuota, restituisce -1. In tutti gli altri casi, restituisce 0.

- pcb_t ***headBlocked**(int *key):
Restituisce il puntatore al pcb del primo processo bloccato sul semaforo, senza deaccordarlo. Se il semaforo non esiste restituisce NULL.

- void **initASL**():
Inizializza la lista dei semdFree in modo da contenere tutti gli elementi della semdTable. Questo metodo viene invocato una volta sola durante l'inizializzazione della struttura dati.

- pcb_t ***removeBlocked**(int *key):
Ritorna il primo PCB dalla coda dei processi bloccati (s_ProcQ) associata al SEMD della ASHT con chiave key.
Se tale descrittore non esiste nella ASHT, restituisce NULL.

Altrimenti, restituisce l’elemento rimosso. Se la coda dei processi bloccati per il semaforo diventa vuota, rimuove il descrittore corrispondente dalla ASHT e lo inserisce nella coda dei descrittori liberi (semdFree).

- void **forallBlocked**(int *key, void fun(pcb_t *pcb, void *),void *arg):
Richiama la funzione fun per ogni processo bloccato sul semaforo identificato da key.

- pcb_t ***outChildBlocked**(pcb_t *p):
Rimuove il PCB puntato da p dalla coda del semaforo su cui è bloccato.

Funzioni ausiliarie

Le funzioni richieste per la gestione della ASHT sono state per la maggior parte realizzate tramite funzioni ausiliarie ricorsive, evitando così l'utilizzo sistematico di variabili globali.
Per semplificare la gestione delle operazioni sulla ASHT sono state implementate 3 funzioni esterne:
-cercaInListaDiTrabocco: per la ricerca di semd con una determinata key nella ASHT;
-insertInListaDiTrabocco: per l'inserimento in una delle liste di trabocco della ASHT;
-removeDaTrabocco: per la rimozione di un semd dalla lista di trabocco a cui appartiene;

Funzione hash

La funzione hash è stata realizzata tenendo conto di due fattori determinanti per una buona distribuzione delle chiavi nella tabella hash:
-la dimensione della tabella hash;
-le possibili ricorrenze delle ultime cifre delle chiavi, che essendo indirizzi di memoria sono quasi sempre multipli di 4;
Per risolvere le possibili variazioni della dimensione della ASHT e non potendola impostare come numero primo, il che avrebbe garantito una buona distribuzione anche con indirizzi di memoria come chiavi, è stato utilizzato un algoritmo in grado di garantire buoni risultati con qualsiasi valore di ASHDSIZE.
Per ovviare al problema delle ricorrenze è stato usato un sistema di mixing dei bit della chiave in modo da ottenere, anche con chiavi molto simili in input, risultati molto diversi in output.

Inserimento e rimozione nelle strutture dati

Ogni semaforo che deve essere aggiunto in una lista di trabocco della ASHT viene inserito in testa. Questo perchè non essendo necessario un ordinamento particolare dei semafori nelle liste, l'inserimento in testa risulta la scelta più efficiente.
Per lo stesso motivo la lista dei semd liberi è realizzata come una pila. Sia l'inserimento(con la funzione ausiliaria "insertInSemdFree") che la rimozione(funzione "removeFromSemdFree") di un elemento dalla lista vengono effettuati in testa.
La coda dei pcb_t bloccati su un semaforo è una coda di priorità. L'inserimento in coda è stato quindi realizzato tramite la funzione "insertProcQ", per la gestione di code di priorità di pcb_t.

Init lista dei semd liberi

La funzione di inizializzazione della lista dei semd liberi, "initASL", esamina l'intera "semdTable" collegando ogni elemento dell'array al successivo modificando il campo s_next. In ogni semd già precedentemente allocato nell'array viene quindi aggiunto il puntatore al successore, creando la lista dei semd liberi. Così facendo non vengono allocati ulteriori semd oltre a quelli della semdTable.