# Relazione fase 1

Per la realizzazione della fase uno è stato necessario costruire le funzioni principali dei process control block che gestiscono i processi inserendoli all'interno di una lista, dove ogni elemento della lista ha al di sotto un albero di processi. In più per poter gestire il controllo di questi processi è stato necessario costituire anche dei semafori che gestiscono questi processi. Per poter fare tutto ciò è stato necessario l'utilizzo di alcune API di linux per il controllo e la gestione di queste strutture (liste , alberi e semafori). Per fare tutto ciò sono state suddivise le strutture in due file che implementano tutte le funzioni: pcb.h con relativo pcb.c e asl.h con anch'esso il relativo asl.c.
Per quanto riguarda le funzioni del pcb queste sono funzioni di gestione della lista di processi e la gestione degli alberi.

**Funzioni per la gestione delle liste di pcb:**

Più nello specifico la lista di processi è una lista bidirezionale con sentinella e questa lista utilizza una particolare struttura chiamata "list_head" che ci permette di creare delle liste e delle funzioni per esse una sola volta, in quanto ogni lista creata essendo basata sulla stessa struttura è possibile riutilizzare le funzioni già scritte in precedenza. Questa particolare struttura verrà utilizzata anche per la realizzazione degli alberi e dei semafori. Più nello specifico le funzioni implementate sono **initPcbs** che grazie alla funzione INIT_LIST_HEAD inizializza correttamente la variabile sentinella e poi con un ciclo di 20 iterazioni (numero di MAXPROC) va ad inserire all'interno della lista in coda tutti i processi; la funzione **freePcb** che prende come parametro un elemento della lista di pcb inserisce in coda l'elemento sentinella quindi andrà a far puntare ,su tutti i puntatori dell'elemento p e quindi anche della sentinella, l'uno con l'altro quindi si avrà che subito dopo l'elemento p la lista sarà vuota perché si tornerà all'elemento sentinella; la funzione **allocPcb** ritornerà un elemento della lista pcb vuoto che è puntato dalla sentinella, se la lista non è vuota. Altrimenti ritornerà NULL; la funzione **mkEmptyProcQ** andrà ad inizializzare la lista dei pcb inizializzando la sentinella e ritornando come output l'elemento puntato da p_next della sentinella. La funzione **emptyProcQ** restituisce vero se l'elemento sentinella punta a se stesso (il che implica anche che la lista è vuota appunto). La funzione **insertProcQ** prendendo in input l'elemento da inserire e l'elemento sentinella inserisce l'elemento secondo due criteri, ovvero se la lista è vuota non ha bisogno di controllare le priorità degli altri elementi della lista quindi quest'ultimo viene solamente inserito in coda, in caso contrario si scorre la lista finchè non si trova un elemento con priorità superiore, quindi si inserisce il nuovo dato tra quello che ha priorità ancora inferiore e quello che ha priorità superiore. La funzione **removeProcQ** rimuove l'elemento con priorità inferiore, quindi l'elemento subito successivo della sentinella, se esiste, in caso contrario restituisce null, che implica che la lista è vuota. Analogamente la funzione **outProcQ** restituisce l'elemento passato come parametro eliminandolo, questo viene fatto scorrendo tutta la lista finchè l'elemento passato in input non viene trovato all'interno della lista, ed a quel punto viene eliminato dalla lista e restituito, in caso non sia presente si ritorna NULL.

**Funzioni per la gestione degli alberi di pcb:**

Per la realizzazione degli alberi di pcb vengono utilizzate le medesime funzioni di gestione delle liste in quanto gli alberi in questa realizzazione sono basati su questa realizzazione. La funzione **emptyChild** restituisce vero se il figlio del

parametro p è soltanto la sentinella , altrimenti restituisce vero se la sentinella punta ad un altro elemento di tipo pcb. La funzione **insertChild** che prende in input il padre e l'elemento da inserire come figlio e va a far puntare al figlio il padre sul campo p_parent e va ad inserire in coda sulla lista di figli del padre, invece la funzione **removeChild** rimuove il primo figlio dell'elemento p in input e lo si restituisce, invece se l'elemento p non ha figli o quest'ultimo non è valorizzato allora restituisce NULL. Infine la funzione **outChild** elimina il figlio inserito in input dalla lista dei figli del padre e mette come padre il valore NULL e restituisce p.

**Funzioni per la gestione della lista di semafori attivi (asl)**

Per la realizzazione e la gestione della lista dei semafori attivi sono state utilizzate anche in questo caso le funzioni relative alle liste dato che anche in questo caso sono state implementate liste di asl grazie alla struttura "list_head", e queste vengono implementate sempre secondo il criterio di lista bidirezionale con sentinella. Avendo fatto questa premessa vengono utilizzate alcune variabili globali che servono per la realizzazione di questa struttura , e tutte queste variabili sono di tipo semd_t. La prima funzione che incontriamo è **getSemd** che ritorna un puntatore ad un tipo semd_t in base ad una chiave. Questo viene fatto scorrendo la lista dei semafori attivi finche non viene trovato un elemento che ha la medesima chiave che viene passata in input, se questo accade viene ritornato quel semaforo, altrimenti viene ritornato NULL come anche nel caso la lista di semafori sia vuota o la chiave sia NULL. La funzione **InsertBlocked** Inserisce un nuovo pcb in coda al semaforo identificato da key. Se tale semaforo è già presente nella ASL allora basta aggiungere il pcb in coda alla lista s_procQ del semaforo. Se, invece, il semaforo non è presente è necessario crearne uno di nuovo. Prima di aggiungere un nuovo semd bisogna controllare che la lista dei semd liberi non sia vuota (se ciò accade la funzione ritorna TRUE). Nel caso in cui il semd sia già presente nella ASL o sia possibile aggiungerlo a quest'ultima, la funzione ritorna FALSE. In tutti gli altri possibili casi di default ritorna FALSE.La funzione **outBlocked** prende in input un pcb da rimuovere da un semaforo. Si compie una ricerca nella ASL utilizzando il campo p_semKey del pcb: se avviene un match, allora si ricerca il pcb dentro la lista s_procQ dei processi bloccati su tale semaforo. Se avviene un match, viene restituito il pcb e quest'ultimo è rimosso dalla coda. In tutti gli altri casi si ritorna NULL. La funzione **headBlocked** prende in input una chiave da ricercare all' interno della ASL: se il semd è attivo allora la funzione ritorna il pcb associato alla testa della lista s_procQ. In tutti gli altri casi la funzione ritorna NULL. La funzione **outChildBlocked** rimuove il pcb dal semd indicato in p_semKey. Inoltre la funzione, ricorsivamente, elimina tutti i pcb che hanno come antenato p. Per fare ciò si fa una previsita dell'albero radicato in p, e mano a mano che si scende si scollega il pcb dal proprio semd e, una volta che le chiamate ricorsive iniziano a chiudersi, i vari pcb vengono scollagati da fratelli e dai propri figli. Infine la funzione **InitASL** inizializza la lista di semafori attivi grazie ad un ciclo che va da 0 a 20 (dimensione di MAXPROC) andando ad inizializzare le sentinelle per ogni processo , viene inizializzata anche la lista dei semafori liberi con relativa sentinella sempre con un ciclo ed infine si inizializza la lista dei semafori attivi.