Andrea Fiorellino
Leonardo Po
Luca Silvestri

Documentazione fase 1 del progetto µPandOS

PCB allocation and deallocation management:

- **void initPcbs():** tramite la funzione freePcb, vengono aggiunti in coda gli elementi della pcbTable (da 1 a MAXPROC) nella lista dei processi liberi;
- **void freePcb(pcb_t *p):** mette l'elemento puntato da p nella lista dei processi liberi;
- **pcb_t *allocPcb():** rimuove il primo elemento dei processi liberi, inizializza tutti i campi e ritorna un puntatore ad esso;

PCB Queue:

- **void mkEmptyProcQ(struct list_head *head):** inizializza una variabile come puntatore alla testa della coda dei processi**;**
- **int emptyProcQ(struct list_head *head)**: se la coda la cui testa è puntata da head è vuota ritorna TRUE, altrimenti FALSE;
- **void insertProcQ(struct list_head *head, pcb_t *p)** :inserisce il PCB puntato da p in fondo alla coda dei processi (puntata da *head);
- **pcb_t *headProcQ(struct list_head *head)**: ritorna NULL se la coda dei processi è vuota, altrimenti il PCB in testa**;**
- **pcb_t *removeProcQ(struct list_head *head)**: rimuove la testa della coda dei processi puntata da *head e ritorna un puntatore dell'elemento in questione; se la lista è vuota ritorna NULL;
- **pcb_t *outProcQ(struct list_head *head, pcb_t *p)**: cerco mediante un for_each il pcb p nella lista puntata da head e lo rimuovo; se lo trovo ritorno p stesso, altrimenti NULL;

PCB Trees

- **int emptyChild(pcb_t *p):** ritorna l'esito della chiamata alla funzione list_empty, alla quale viene passato come parametro l'indirizzo del list_head p_child di p;
- **void insertChild(pcb_t *prnt, pcb_t *p):** si assegna prnt al puntatore p_parent di p. Dopo si aggiunge p alla lista dei fratelli, tramite list_add (se non ci sono altri figli) e list_add_tail (per rispettare la FIFOness), alle quali viene passato come parametro gli indirizzi del list_head p_sib di p e del list_head p_child di prnt (p diventa fratello dei figli di prnt e quindi figlio di p).
- **pcb_t *removeChild(pcb_t *p):** il controllo sulla presenza o meno di figli avviene con la funzione emptyChild. Se ci sono figli, si sceglie il primo figlio tramite la macro container_of, chiamata sull'elemento successivo al list_head p_child. In seguito il figlio viene eliminato tramite la funzione list_del e viene troncato il legame con il padre, assegnando il valore NULL il puntatore p_parent del figlio.
- **pcb_t *outChild(pcb_t *p):** se p ha un padre, rimuovo p dalla lista dei suoi fratelli chiamando list_del a cui passo come parametro l'indirizzo di p_sib di p, in seguito rimuovo il legame con il padre assegnando NULL al puntatore p_parent di p.

Message allocation and deallocation management

- **void freeMsg(msg_t *m):** Inserisce l'elemento puntato da m in testa alla lista dei messaggi.
- **msg_t *allocMsg():** Ritorna NULL se la lista dei messaggi è vuota. Altrimenti rimuove un elemento dalla testa, imposta a 0 la variabile m_payload di ogni messaggio presente nell'array msgTable e ritorna un puntatore all'elemento rimosso.
- **initMsgs():** Inserisce gli elementi presenti nell'array msgTable in coda alla lista dei messaggi.

Message

- **void mkEmptyMessageQ(struct list_head *head):** Inizializza una una lista di messaggi vuota.
- **int emptyMessageQ(struct list_head *head):** Ritorna 1 se la lista puntata da head è vuota, altrimenti 0.

- **void insertMessage(struct list_head *head, msg_t *m):** Inserisce il messaggio puntato da m in coda alla lista puntata da head.
- **void pushMessage(struct list_head *head, msg_t *m):** Inserisce il messaggio puntato da m in testa alla lista puntata da head.
- **msg_t *popMessage(struct list_head *head, pcb_t *p_ptr):** Rimuove il primo messaggio trovato nella lista puntata da head che è stato inviato dal thread p_ptr.
  Se p_ptr è NULL, ritorna il primo messaggio in coda.
  Se head è vuota o se non viene trovato alcun elemento mandato dal thread p_ptr, ritorna null.
- **msg_t *headMessage(struct list_head *head):** Se la lista puntata da head è vuota ritorna NULL, altrimenti ritorna il messaggio in testa ad essa.

Membri:

Fiorellino Andrea, matricola: 0001089150, email: andrea.fiorellino@studio.unibo.it
Po Leonardo, matricola: 0001069156, email: leonardo.po@studio.unibo.it
Silvestri Luca, matricola: 0001080369, email: luca.silvestri9@studio.unibo.it