Prova pratica di Calcolatori Elettronici (nucleo v6.*)

C.d.L. in Ingegneria Informatica, Ordinamento DM 270

19 gennaio 2016

1. Due processi possono comunicare tramite una `pipe`, un canale con una estremità di scrittura e una di lettura attraverso il quale viaggia una sequenza di caratteri. I caratteri inviati dall'estremità di scrittura possono essere letti dall'estremità di lettura.

   Per realizzare le `pipe` aggiungiamo le seguenti primitive (abortiscono il processo in caso di errore):

   - `natl inipipe()` (tipo 0x5c, già realizzata): Crea una nuova pipe e ne restituisce l'identificatore (`0xFFFFFFFF` se non è stato possibile creare una nuova pipe).
   - `void writepipe(natl p, char *buf, natl n)` (tipo 0x5d, da realizzare): Invia `n` caratteri dal buffer `buf` sulla pipe di identificatore `p`. È un errore se la pipe `p` non esiste.
   - `void readpipe(natl p, char *buf, natl n)` (tipo 0x53, da realizzare): Riceve `n` caratteri dal dalla pipe di identificatore `p` e li scrive nel buffer `buf`. È un errore se la pipe `p` non esiste.

   Previdamo un tipo di `pipe` senza un buffer interno. Per la `writepipe`, questo vuol dire che i caratteri possono essere trasferiti solo se un altro processo è pronto a riceverli tramite una `readpipe`, altrimenti la primitiva deve attendere. Inoltre, il processo che ha invocato la `readpipe` potrebbe aver chiesto meno caratteri di quelli da inviare: in questo caso si devono inviare i caratteri possibili e continuare ad attendere; questa operazione potrebbe ripetersi più volte fino a quando tutti i caratteri non sono stati trasferiti. Analoghe considerazioni valgono per la `readpipe`.

   Per semplicità non trattiamo i casi in cui più di un processo voglia accedere alla stessa estremità della stessa pipe. Inoltre, assumiamo che i buffer passati alla `readpipe` e alla `writepipe` appartengano allo spazio utente comune.

   Per descrivere una `pipe` aggiungiamo al nucleo la seguente struttura dati:

   ```
   struct des_pipe {
       des_proc* w_wait;
       char *w_buf;
       natl w_pending;

       des_proc* r_wait;
       char *r_buf;
       natl r_pending;
   };
   ```

   Il campo `w_wait` punta all'eventuale processo in attesa di poter completare una `writepipe`; vale `NULL` se non ve ne sono. Se `w_wait` non è nullo, `w_buf` punta al buffer contenente i caratteri ancora da trasferire e `w_pending` ne indica il numero. I campi `r_wait`, `r_buf` e `r_pending` svolgono un ruolo analogo per i processi in attesa di completare una `readpipe`.

   Modificare i file `sistema.cpp` e `sistema.s` in modo da realizzare le primitive mancanti.