Esercizio sulle Code di Messaggi

- In un moderno SO, sono costantemente attivi alcuni processi di sistema (normalmente lanciati durante la fase di bootstrap) che forniscono determinati servizi agli utenti che ne fanno richiesta.
- In Unix, questi processi vengono detti "demoni" o "**deamon**", e hanno di solito un nome formato dalla parola usata per indicare il servizio che forniscono seguito dalla lettera "d".
- Se visualizzate i processi che girano su un sistema Unix, potete identificare i demoni presenti. Di alcuni, possono essere attive più istanze, perché più utenti stanno usando contemporaneamente il servizio fornito dal demone corrispondente.

Esercizio sulle Code di Messaggi

- Ad esempio, sui server del dipartimento troviamo:
  - **httpd:** il demone di gestione del web server. HTTP è il protocollo di comunicazione usato tra web server e browsers (Netscape, Explorer) per inviare e ricevere pagine web
  - **telnetd:** il demone di gestione delle connessioni telnet, di solito provenienti da un altra macchina
  - **lpd:** il demone di gestione delle stampanti, e quindi dei comandi di stampa inviati sulla macchina
  - **ftpd:** il demone di gestione delle operazioni di ftp(file transfer protocol): trasferimento di file tra macchine diverse
  - **sshd:** il demone di gestione delle connessioni remote attraverso ssh (secure shell).

Esercizio sulle Code di Messaggi

- Implementate un semplice demone (che chiameremo "*cuoco*") che rimane in attesa attesa su una coda di messaggi di richieste di preparare un determinato piatto (ad esempio un panino, o un arrosto, o una pasta) da parte dei clienti.
- Quando arriva sulla coda un messaggio di richiesta di un piatto, il *cuoco* si forka, e mentre il *cuoco figlio* prepara il piatto richiesto, il *cuoco* (ossia il padre) si rimette immediatamente in attesa di altre richieste in arrivo sulla coda. In un dato istante, ci possono quindi essere più *cuochi figli* che stanno preparando i piatti richiesti.

Esercizio sulle Code di Messaggi

- Quando un cliente ha inserito nella coda la richiesta di un certo piatto (ad esempio un panino), rimane in attesa sulla stessa coda che il piatto richiesto gli venga consegnato.
- Quando un *cuoco figlio* ha terminato di preparare il piatto richiestogli, lo invia sulla coda al cliente che lo aveva ordinato.
- Ovviamente, ogni cliente dovrà ricevere, sulla coda, il piatto che ha ordinato, e non quello ordinato da un altro cliente anch'egli in attesa del proprio piatto.

Esercizio sulle Code di Messaggi

- Per inviare la richiesta di un certo piatto, predisponete un comando *prepara* (di fatto, un eseguibile C) che permette di inviare al cuoco sulla coda di comunicazione una richiesta in questo modo:

```
$> prepara panino
```

oppure:

```
$> prepara gelato
```

e così via.

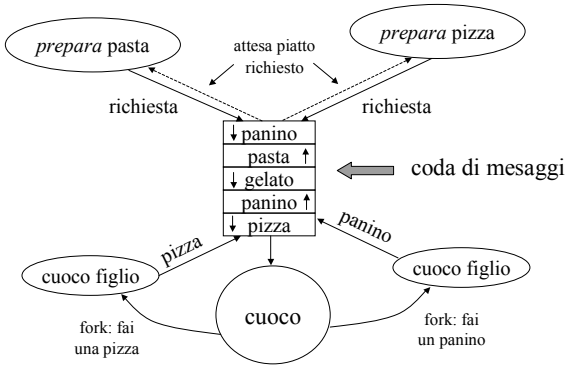
Esercizio sulle Code di Messaggi

- Cosa deve fare *prepara*:
  - invia un messaggio al *cuoco* contenente il nome del piatto XXX passato come argomento in input
  - si mette in attesa sulla stessa coda della risposta proveniente dal *cuoco figlio* che ha gestito la richiesta.
  - La risposta deve contenere il nome del piatto YYY preparato dal *cuoco figlio*.
  - *prepara* stampa in ouput il nome del piatto ricevuto YYY e termina.
  - Ovviamente, deve essere XXX = YYY

Esercizio sulle Code di Messaggi

- Cosa deve fare il *cuoco:* appena lanciato (in background) preleva una coda di messaggi su cui avverranno tutte le comunicazioni, e poi si mette ad eseguire un loop infinito (è un demone) in cui:
  - rimane in attesa sulla coda di richieste di preparazione di piatti.
  - quando riceve un messaggio contenente il piatto XXX si forka, e mentre il *figlio cuoco* prepara XXX il *cuoco* si rimette in attesa della prossima richiesta.
- Il *cuoco figlio* che deve preparare XXX:
  - aspetta un tempo random da 1 a 5 secondi
  - invia sulla coda, al processo *prepara* che ne aveva fatto richiesta, il piatto XXX, e termina.

Esercizio sulle Code di Messaggi



Esercizio sulle Code di Messaggi

- **ATTENZIONE**: qual è il punto "delicato" dell'esercizio? C'e' una sola coda di comunicazione, e ogni istanza di *prepara* deve ricevere un messaggio contentente il nome del piatto che ha ordinato. Non deve invece intercettare erroneamente messaggi destinati ad altre istanze di *prepara* che hanno chiesto piatti diversi.
- Naturalmente, questo può verificarsi solo se vengono lanciati contemporaneamente più comandi *prepara* (presumibilmente da utenti diversi e con piatti diversi), di modo che ci sono più *cuochi figli* contemporaneamente attivi che preparano un piatto e inviano sulla coda il messaggio contentente il nome di quel piatto.

Esercizio sulle Code di Messaggi

- **SUGGERIMENTO**: nel messaggio inviato da una istanza di *prepara*, oltre al nome del piatto richiesto, deve essere presente qualcosa che identifichi univocamente quella istanza, e che permetta al *cuoco figlio* che serve la richiesta, di inviare un messaggio (contente il piatto preparato) **che possa essere ricevuto solo** dall' istanza di prepara che ne aveva fatto richiesta

Prelievo e uso di una risorsa da parte di più processi

- Quando viene lanciato un comando *prepara*, come fa il processo a sapere l'identificatore della coda su cui comunicare col *cuoco*?
- Un modo semplice e' usare un file *id_file* per contere l'identificativo della coda, scritto dal *cuoco* quando questo preleva la coda.
- Quando il comando *prepara* viene lanciato, legge in *id_file* l'identificativo della coda su cui inviare e ricevere i messaggi

Prelievo e uso di una risorsa da parte di più processi

- C'e' pero' un modo piu' elegante per far usare ad un insieme di processi una determinata coda (o altra risorsa):

  msgid = msgget(KEY, IPC_CREAT | 0666)
- KEY e' un numero intero positivo concordato da tutti i programmi che vogliono usare la coda
- Alla prima chiamata della msgget con KEY, viene creata una nuova coda e il suo identificatore msgid associato biunivocamente a KEY.
- successivamente, qualsiasi altro programma che chiama msgget con lo stesso valore di KEY riceve il msgid associato a KEY.

Prelievo e uso di una risorsa da parte di più processi

- ad esempio, in un file mydef.h possiamo mettere:

```
#define MIA_CODA 12345
```

- e in ogni programma che deve usare MIA_CODA:

```
#include mydef.h
..... .....
msgid = msgget(MIA_CODA, IPC_CREAT|0666);
```

- Se un processo (come nel caso del comando *prepara*) sa che la coda è già stata allocata (dal *cuoco*, nel nostro caso) allora può semplicemente usare:

```
msgid = msgget(MIA_CODA, 0);
```

Prelievo e uso di una risorsa da parte di più processi

- Notate: se 2 o piu' processi condividono lo stesso padre, allora come valore di KEY si può usare il PID del padre:

```
msgid = msgget(getppid(), IPC_CREAT|0666);
```

Alcuni consigli:

- Tenete *prepara* e *cuoco* in un unica cartella, per semplicità.
- Prima di usare *prepara*, *cuoco* deve già essere stato lanciato, in background.
- Quando avete finito di provare i programmi, ricordatevi di terminare il demone *cuoco* e di rimuovere la coda di messaggi allocata.
- Potete prevedere dentro *cuoco* una procedura di rimozione della coda e terminazione del demone, da eseguire alla ricezione di un opportuno segnale inviato da terminale mediante kill. (esempio: “kill –SIGUSR1 pid_del_cuoco”)

Alcuni consigli:

- Come si può verificare che i programmi cuoco e prepara stiano funzionando correttamente?
- Potete inserire in *prepara* due stampe di diagnostica, in questo modo:

```
printf("\n %d: ordino un %s \n",getpid(), argv[1]);
msgsnd(...);          // invio richiesta al cuoco
msgrcv(...);          // ricevo il piatto dal cuoco figlio
printf("\n %d: mi hanno portato un %s \n, getpid(), msgp->mtext);
```

Alcuni consigli:

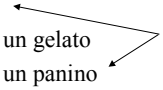
- Stampando, insieme al piatto ordinato/ricevuto, il PID del processo *prepara* che ha generato la richiesta, si può verificare la corrispondenza tra quanto ordinato e quanto ricevuto da una data istanza di *prepara.* Ad esempio:

```
$>prepara panino &
$>prepara gelato &
543: ordino un panino
678: ordino un gelato
678: mi hanno portato un gelato
543: mi hanno portato un panino
```

Alcuni consigli:

- Ma se si verificasse un output del tipo:

```
$>prepara panino &
$>prepara gelato &
543: ordino un panino
678: ordino un gelato
543: mi hanno portato un gelato
678: mi hanno portato un panino
```



- Allora c'è qualcosa che non va...

Alcuni consigli:

- Preparate quindi uno script "ordinazioni" con una sequenza di chiamate a *prepara.* Ad esempio:

```
prepara panino &
prepara gelato &
prepara arrosto &
prepara gelato &
prepara risotto &
prepara dessert &
```

← ordinazioni

- eseguitelo e controllate che ogni istanza di prepara riceva il piatto che ha effettivamente ordinato.

Osservazione:

- Il metodo di controllo proposto vi da la sicurezza che i programmi scritti funzionino correttamente?
- Riuscite a pensare ad un metodo di controllo migliore?

Osservazione sull'uso di printf:

- Come abbiamo visto, inserendo delle printf in un programma C, è possibile controllarne il funzionamento.
- Tuttavia, il sistema di I/O di Unix è gestito in maniera bufferizzata, per cui, una stampa non viene visualizzata immediatamente dopo che è stata eseguita.
- Addirittura, se un programma esegue una printf e poi termina, è possibile che quella printf non venga visualizzata sul terminale, anche se è stata regolarmente eseguita.

Osservazione sull'uso di printf:

- Per ovviare a possibili inconvenienti di questo tipo, è consigliabile inserire, dopo ogni printf (o dopo un gruppo di printf consecutive) l'istruzione fflush:

```
printf("\n ciao! ");
printf("Come stai?\n");
fflush(stdout);
....
```

- In questo modo, se una printf è stata eseguita dalla CPU, l'output sarà certamente visualizzato sul terminale