PROGETTO SISTEMI OPERATIVI – LINUX SYSTEM CALL 2021/2022

- **Prenotazione slot nel calendario esami-orali: 31/05/2022**
- **Consegna: entro 08/06/2022 23:59**
- **Date esami orali: 13/06/2022 – 24/06/2022 (Lunedì-Venerdì)**

Descrizione generale

Si realizzi un'applicazione che implementa l'architettura di Figura 1.

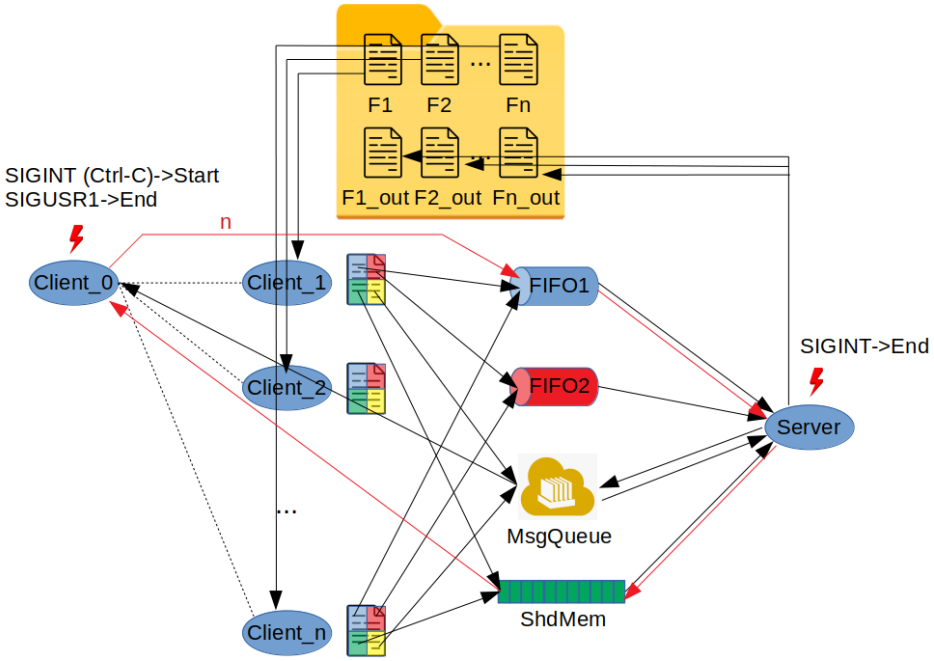

Figura 1: architettura del progetto

Un processo Server genera due FIFO (FIFO1 e FIFO2), una coda di messaggi (MsgQueue), un segmento di memoria condivisa (ShdMem) ed un set di semafori per gestire la concorrenza su alcuni di questi strumenti di comunicazione.
Successivamente il processo Server si mette in ascolto su FIFO1.
(Nota: Le IPC potranno contenere al massimo 50 messaggi. Hint: per quanto riguarda la ShdMem usare un vettore di supporto per identificare le locazioni di memoria libere.)

Processo Client_0

Quando il processo Client_0 viene avviato (con il comando *./Client_0 <HOME>/myDir/*, dove <HOME> e' la cartella home dell'utente, cioe' il valore contenuto nella variabile d'ambiente HOME), crea una maschera che gli consente di ricevere solo i segnali SIGINT e SIGUSR1 e si mette in attesa di uno di questi segnali. Alla ricezione del segnale SIGUSR1 il processo Client_0 termina. Alla ricezione del segnale SIGINT (Ctrl-C da tastiera) il processo Client_0:

- blocca tutti i segnali (compresi SIGUSR1 e SIGINT) modificando la maschera
- imposta la sua directory corrente ad un path passato da linea di comando all'avvio del programma,
- saluta l'utente stampando a video la stringa "Ciao USER, ora inizio l'invio dei file contenuti in CURRDIR", dove USER e' il nome utente e CURRDIR è la directory corrente,
- accede a tutte le cartelle presenti nella sua directory corrente e carica in memoria tutti i percorsi dei file il cui nome inizia con la stringa "sendme_" e la dimensione è inferiore a 4KByte

- determina il numero "*n"* di questi file e lo invia tramite FIFO1 al server (percorso rosso in Fig 1), poi si mette in attesa di conferma dal server su ShrMem
- una volta ricevuta conferma dal server, per ciascun file con nome che inizia con la stringa "sendme_" genera un processo figlio Client_i (linee tratteggiate in Figura 1) il quale esegue le seguenti operazioni (*Nota: l'elaborato dovrà funzionare con al più 100 file "sendme", ciascuno con dimensione massima di 4KByte*).
  - apre il file,
  - determina il numero di caratteri totali,
  - divide il file in quattro parti contenenti lo stesso numero di caratteri (se il numero di caratteri non e' divisibile per 4, l'ultima parte conterrà un numero inferiore di caratteri),
  - prepara i quattro messaggi per l'invio,
  - si blocca su un semaforo fino a quando tutti i client Client_1 - Client_N sono arrivati a questo punto dell'esecuzione. Hint: Usare semop() per attendere che il semaforo arrivi a zero,
  - quando il semaforo consente al Client_i di proseguire esso invia il primo messaggio a FIFO1, il seconda a FIFO2, il terza a MsgQueue ed il quarto a ShdMem (frecce nere in Figura 1); all'interno dei messaggi, Client_i invia anche il proprio PID ed il nome del file "*sendme_"* (con percorso completo),
  - chiude il file,
  - termina.
- (Client_0) si mette in attesa sulla MsgQueue di un messaggio da parte del server che lo informa che tutti i file di output sono stati creati dal server stesso e che il server ha concluso le sue operazioni.
- una volta ricevuto tale messaggio Client_0 sblocca i segnali SIGINT e SIGUSR1 e si rimette in attesa di ricevere uno dei due segnali.

Si noti che la comunicazione tramite i quattro canali (i.e., FIFO1, FIFO2, MsgQueue, e Shdmem) avviene in modo concorrente tra tutti i processi Client_i, quindi, dove necessario, si devono implementare dei meccanismi per gestire tale concorrenza. In particolare, ogni processo Client_i accede a ciascun canale di comunicazione ed inserisce al suo interno il relativo messaggio + informazioni di identificazione. I messaggi dei vari processi Client_i si possono mescolare all'interno dei canali di comunicazione. Il server deve gestire il riordino delle varie parti in modo da ottenere il messaggio completo inviato da ciascun processo Client_i.

Processo Server

Il processo Server, alla ricezione da FIFO1 del numero "n" di file trasmessi dal client (percorso rosso in Figura 1)

- memorizza tale numero
- scrive un messaggio di conferma su ShdMem (percorso rosso in Figura 1)
- si mette in ricezione ciclicamente su ciascuno dei quattro canali (frecce nere in Figura 1)

Alla ricezione dei messaggi dai vari canali esegue le seguenti operazioni:

- memorizza il PID del processo mittente, il nome del file con percorso completo ed il pezzo di file trasmesso
- una volta ricevute tutte e quattro le parti di un file le riunisce nell'ordine corretto e le salva in un file di testo in cui ognuna delle quattro parti e' separata dalla successiva da una riga bianca (carattere newline) ed ha l'intestazione "[Parte j, del file NOMEFILE, spedita dal processo PID tramite CANALE]" (vedere esempio sotto), dove j è un numero da 1 a 4 in base alla parte del file, NOMEFILE è il nome del file di origine compreso di percorso completo, PID è il PID del processo mittente e CANALE è il canale di comunicazione (uno tra FIFO1, FIFO2, MsgQueue e ShdMem). Il file verrà chiamato con lo stesso nome (e percorso) del file originale ma con l'aggiunta del postfisso "_out"

- quando ha ricevuto e salvato tutti i file invia un messaggio di terminazione sulla coda di messaggi, in modo che possa essere riconosciuto da Client_0 come messaggio di conclusione lavori
- si rimette in attesa su FIFO 1 di un nuovo valore n

Alla ricezione del comando SIGINT (Ctrl-C da terminale) il processo Server rimuove tutte le IPC e termina.

Esempio

Se nella cartella <HOME>*/myDir* ci sono 3 file contenti i seguenti testi:

- sendme_File1: AABBCCDD
- sendme_File2: EEFFGGHH
- sendme_File3: IILLMMNN

Client_0 inizialmente blocca i segnali e si blocca fino a quando non viene sbloccato alla ricezione di SIGINT. Una volta sbloccato invia il numero 3 a Server tramite FIFO1 e Server risponde con un messaggio di conferma su ShdMem. Quando Client_0 riceve il messaggio, genera tre processi: Client_1 per l'invio di sendme_File1, Client_2 per l'invio di sendme_File2, Client_3 per l'invio di sendme_File3.
I tre Client_i effettuano la comunicazione nel seguente modo (*Nota: l'invio dei messaggi non è puramente sequenziale*):

- Client_1 (avente per esempio pid1=10):
  - invia messaggio [AA, 10, <HOME>/myDir/sendme_File1] su FIFO1,
  - invia messaggio [BB, 10, <HOME>/myDir/sendme_File1] su FIFO2,
  - invia messaggio [CC, 10, <HOME>/myDir/sendme_File1] su msgQueue, e
  - invia messaggio [DD, 10, <HOME>/myDir/sendme_File1] su ShdMem.
- Client_2 (avente per esempio pid1=11):
  - invia messaggio [EE, 11, <HOME>/myDir/sendme_File2] su FIFO1,
  - invia messaggio [FF, 11, <HOME>/myDir/sendme_File2] su FIFO2,
  - invia messaggio [GG, 11, <HOME>/myDir/sendme_File2] su msgQueue, e
  - invia messaggio [HH, 11, <HOME>/myDir/sendme_File2] su ShdMem.
- Client_3 (avente per esempio pid1=12):
  - invia messaggio [II, 12, <HOME>/myDir/sendme_File3] su FIFO1,
  - invia messaggio [LL, 12, <HOME>/myDir/sendme_File3] su FIFO2,
  - invia messaggio [MM, 12, <HOME>/myDir/sendme_File3] su msgQueue, e
  - invia messaggio [NN, 12, <HOME>/myDir/sendme_File3] su ShdMem.

Il server riceve i messaggi, li ricompone e genera i seguenti file di output:

- sendme_File1_out:
  [Parte 1, del file <HOME>/myDir/sendme_File1, spedita dal processo 10 tramite FIFO1]
  AA

  [Parte 2, del file <HOME>/myDir/sendme_File1, spedita dal processo 10 tramite FIFO2]
  BB

  [Parte 3, del file <HOME>/myDir/sendme_File1, spedita dal processo 10 tramite MsgQueue]
  CC

  [Parte 4, del file <HOME>/myDir/sendme_File1, spedita dal processo 10 tramite ShdMem]
  DD

- sendme_File2_out:
  [Parte 1, del file <HOME>/myDir/sendme_File2, spedita dal processo 11 tramite FIFO1]
  EE

  [Parte 2, del file <HOME>/myDir/sendme_File2, spedita dal processo 11 tramite FIFO2]
  FF

  [Parte 3, del file <HOME>/myDir/sendme_File2, spedita dal processo 11 tramite MsgQueue]
  GG

  [Parte 4, del file <HOME>/myDir/sendme_File2, spedita dal processo 11 tramite ShdMem]
  HH

- sendme_File3_out:
  [Parte 1, del file <HOME>/myDir/sendme_File3, spedita dal processo 12 tramite FIFO1]
  II

  [Parte 2, del file <HOME>/myDir/sendme_File3, spedita dal processo 12 tramite FIFO2]
  LL

  [Parte 3, del file <HOME>/myDir/sendme_File3, spedita dal processo 12 tramite MsgQueue]
  MM

  [Parte 4, del file <HOME>/myDir/sendme_File3, spedita dal processo 12 tramite ShdMem]
  NN

Una volta terminata la scrittura di tutti i messaggi manda un messaggio di conferma a Client_0 tramite MsgQueue e si rimette in ascolto su FIFO1.

Quando Client_0 riceve il messaggio di conferma, sblocca i segnali SIGINT e SIGUSR1 e si rimette in attesa di tali segnali per effettuare altre comunicazioni o terminare.

Esecuzione programmi e relativi input/output
Il progetto deve essere **compilato** con il comando *make* per mezzo del *makefile* fornito nel template del progetto.

L'**esecuzione** del progetto deve avvenire per mezzo dei seguenti comandi che **devono poter essere eseguiti dall'interno della cartella *~/sistemi_operativi/system_call/***:

./Server
./Client_0 <HOME>/myDir/

Gli **input** sono i file di testo con nome avente prefisso *sendme_* contenuti nella cartella <HOME>/myDir/ o in sue sottocartelle. Si noti che il codice deve essere in grado di trovare i file anche nelle sottocartelle di <HOME>/myDir/ e deve saper distinguere i file in base al loro prefisso.

Gli **output** sono i file di testo con suffisso *_out* generati da Server e memorizzati nella cartella <HOME>/myDir/ o in sue sottocartelle. Per ciascun file con prefisso *sendme_* si deve generare un rispettivo file con suffisso *_out* nella stessa cartella.

Altre informazioni
Tutto ciò non espressamente specificato nel testo del progetto è a scelta dello studente.

È vietato l'uso di funzioni C per la gestione del file system (e.g. fopen). Il progetto deve funzionare su sistema operativo Ubuntu 18 e Repl, rispettare il template fornito, ed essere compilabile con il comando *make.*

L'ammissione all'esame orale è possibile solo se **rispettata la data di consegna dell'elaborato nel secondo semestre e se l'elaborato compila ed esegue correttamente.**

Si consiglia di svolgere il progetto a gruppi di tre persone ma si possono creare gruppi anche più piccoli se necessario. Ogni gruppo dovrà consegnare un solo progetto.

- La consegna di questo elaborato permette di ottenere un punteggio massimo di 27/30.
- La consegna dell'esercitazione su /MentOS/scheduler_algorithm.c permette di ottenere un punteggio massimo aggiuntivo di 2/30.
- La consegna dell'esercitazione su MentOS/deadlock_più_file.c (vedi sotto) permette di ottenere un punteggio massimo aggiuntivo di 3/30.

La votazione massima è quindi 27+2+3=32 corrispondente a 30 e lode.

IMPORTANTE: *Tutti i componenti del gruppo verranno interrogati su tutto il progetto, non su singole parti del progetto "su cui loro hanno lavorato". Il progetto è alla base del interrogazione ma non ne definisce il voto.*

Consegna elaborato e-learning
Si richiede di rispettare la seguente struttura di cartelle per la consegna dell'elaborato:
/sistemi_operativi
    /system_call/ (tutti i file del template che vi andremmo a fornire)
    /MentOS/deadlock_prevention.c (da consegnare nel secondo semestre)
    /MentOS/smart_sem_user.c
    /MentOS/syscall.c
    /MentOS/syscall_types.h

La directory sistemi_operativi deve essere compressa in un archivio di nome <matricola1_matricola2_matricola3>_sistemi_operativi.tar.gz. L'archivio deve essere creato con il comando tar (no programmi esterni).

In caso di gruppi con meno di 3 studenti si mettano nel nome del file solo le matricole degli studenti che hanno partecipato. L'archivio tar.gz deve essere caricato nell'apposita sezione sul sito di e-learning.

Nota bene 1*: gli elaborati dovranno passare un test preliminare di compilazione ed esecuzione automatica. Gli elaborati che non passano tale test perché non rispettano le specifiche non saranno considerati validi per l'ammissione all'orale, quindi si chiede di seguire fedelmente le indicazioni ed il formato del template.*

***Nota bene 2**: Il gruppo puo essere diverso da quello creato al primo semestre. Esempio, un gruppo composto da due persone che ha consegnato la prima esercitazione al primo semestre, puo presentarsi al secondo semestre con un componente aggiuntivo.*

***Nota bene 3**: le FIFO possono essere create in qualsiasi punto del filesystem per cui il vostro utente ha i permessi di scrittura. Noi consigliamo di generare le FIFO a partire dalla posizione dove viene mandato in esecuzione il vostro eseguibile. Su replit dovete tenere a mente che la vostra home si trova al interno di "/home/runner/".*