PROGETTO SISTEMI OPERATIVI – LINUX SYSTEM CALL 2020/2021

- **Prima consegna elaborato: 28/02/2021**
- **Seconda consegna elaborato: deadline 06/06/2021 ore 23.59**
- **Prenotazione slot nel calendario esami-orali: deadline 31/05/2021 ore 23.59**
- **Date esami orali: 14/06/2021-25/06/2021 (Lunedi - Venerdì)**

Descrizione generale

Si vuole realizzare una applicazione che simuli varie tipologie di comunicazione sequenziale tra processi con un processo "disturbatore" che modifica la sequenza pre-specificata di comunicazione inviando segnali ai processi che eseguono la comunicazione.

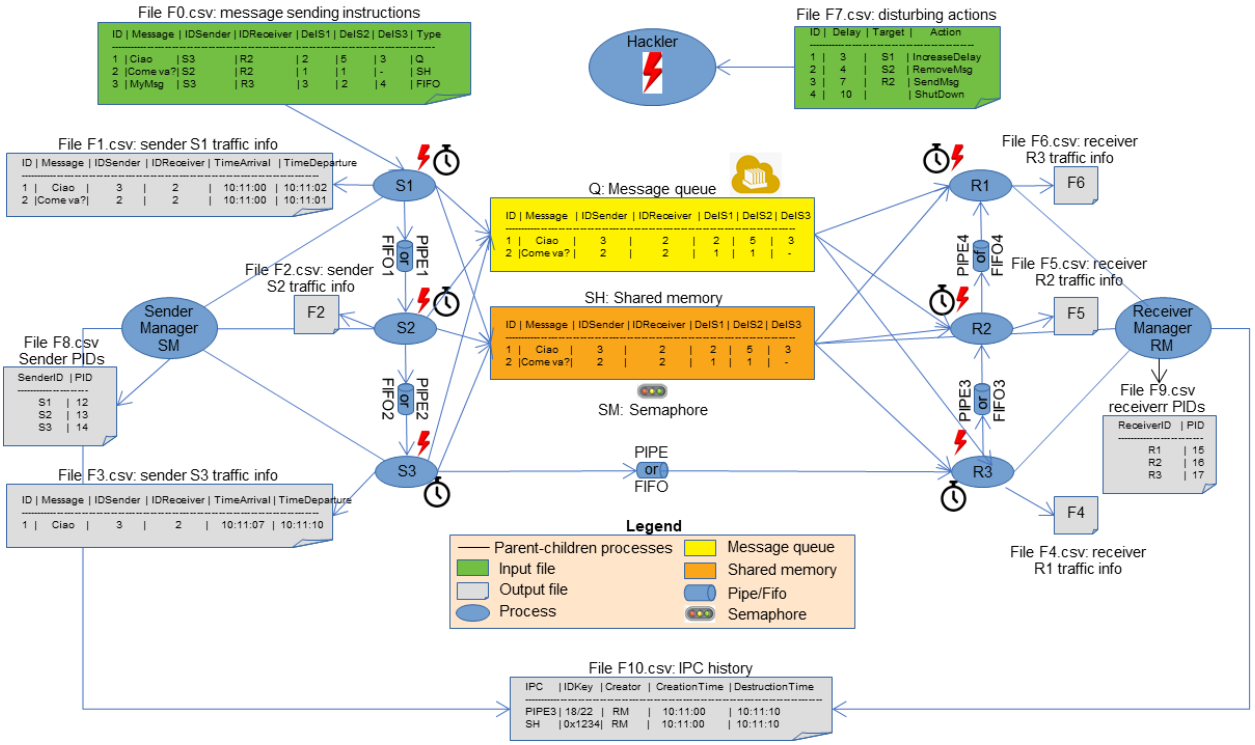


Figura 1: architettura del sistema da sviluppare

All'avvio dell'applicazione un processo **SenderManager** (SM) genera tre processi figlio **Sender** (S1, S2, S3) e collega S1 a S2, ed S2 a S3 tramite due PIPE (se lo studente non riesce a gestire la comunicazione con PIPE può utilizzare delle FIFO, la scelta dovrà essere motivata in sede d'esame orale), rispettivamente PIPE1/FIFO1 e PIPE2/FIFO2 in Figura 1. Il processo SM genera anche una **coda di messaggi** (i.e., Q), un'area di **memoria condivisa** (i.e., SH), un **semaforo** (i.e., SM) per la sincronizzazione degli accessi alla memoria SH, ed una **PIPE/FIFO**. I processi S1, S2 ed S3 possono inviare messaggi alla coda di messaggi e scrivere sull'area di memoria condivisa. Il processo S3 può inoltre anche inviare dati al receiver 3 tramite una FIFO. Il processo SM salva su file **F8.csv** il mapping tra nome processo figlio (e.g., S1) ed il relativo PID (il formato di tale file è definito nel file F8.csv allegato al template del codice).

Il processo **ReceiverManager** (RM) genera tre processi figlio **Receiver** (R1, R2, R3) e collega R3 a R2 ed R2 a R1 tramite due pipe rispettivamente PIPE3 e PIPE4 (se lo studente non riesce a gestire la comunicazione con PIPE può utilizzare delle FIFO, la scelta dovrà essere motivata in sede d'esame orale). Tutti e tre i processi R1, R2 ed R3 devono essere in grado di leggere dalla coda di messaggi e

dalla memoria condivisa. Il processo R3 può anche leggere dalla PIPE/FIFO. Il processo RM salva su file **F9.csv** il mapping tra nome processo figlio (e.g., R1) ed il relativo PID (il formato di tale file è definito nel file F9.csv allegato al template del codice).

Assieme ad SM e RM, all'avvio, viene anche generato un processo **Hackler** (disturbatore) in grado di modificare il regolare svolgimento delle operazioni inviando dei segnali ai processi sender S1, S2 ed S3 o ai processi receiver R1, R2 ed R3, come definito in dettaglio nel seguito.

Una volta che SM ha generato tutti i processi e gli strumenti di comunicazione, il processo S1 legge dal file **F0.csv** (allegato al template del codice) una lista di informazioni nel seguente formato:

ID | *Message* | *IDSender* | *IDReceiver* | *DelS1* | *DelS2* | *DelS3* | *Type*

dove:

- ID: identificativo del messaggio (e.g., 1)
- Message: testo del messaggio (e.g., "Ciao ciao") di dimensione massima di 50 caratteri
- IdSender: identificativo del processo sender che dovrà inviare il messaggio (i.e., S1, S2 o S3)
- IdReceiver: identificativo del processo receiver che dovrà ricevere il messaggio (i.e., R1, R2 o R3)
- DelS1: tempo (in secondi) in cui il messaggio dovrà rimanere in attesa nella memoria del processo S1 (tale memoria non e' condivisa ma una apposita struttura dati del processo S1)
- DelS2: tempo (in secondi) in cui il messaggio dovrà rimanere in attesa nella memoria del processo S2 (tale memoria non e' condivisa ma una apposita struttura dati del processo S2)
- DelS3: tempo (in secondi) in cui il messaggio dovrà rimanere in attesa nella memoria del processo S3 (tale memoria non e' condivisa ma una apposita struttura dati del processo S3)
- Type: tipo di comunicazione tra sender e receiver (i.e., Q, SH, FIFO)

Ogni riga rappresenta un messaggio che uno specifico sender (IDSender) deve inviare ad uno specifico receiver (IDReceiver) per mezzo di una specifica modalità di comunicazione (Type).
Il processo S1 genera una struttura (**struct message in c**) per ciascun messaggio (riga letto da F0) contenente tutte le informazioni utili alla comunicazione del messaggio stesso.
Poi fa partire un timer per ciascun messaggio in modo da far attendere la partenza di ciascun messaggio del relativo numero di secondi indicato nel campo DelS1 del messaggio.

La gestione dei messaggi richiesta **non è sequenziale.** In particolare, il sender S1 legge all'inizio tutti i messaggi (supponiamo ad esempio che siano due, uno con attesa di 5 secondi ed uno con attesa di 7 secondi), li salva internamente ed inizia ad attendere che il primo messaggio possa essere inviato. Quando tale attesa è terminata (dopo 5 secondi nel caso d'esempio) invia il relativo messaggio e fa partire l'attesa per il secondo messaggio (relativamente al tempo rimanente che sarà di 2 secondi nel caso d'esempio). Questo meccanismo di attesa asincrona può essere gestito dallo studente come preferisce. Si può ad esempio gestire un allarme che viene aggiornato opportunamente oppure si possono generare nuovi processi per gestire le attese dei vari messaggi in modo indipendente. Facciamo notare che durante l'attesa di un Sender/Receiver potrebbero arrivare altri messaggi che vanno inserite nella coda di attesa in modo che possano essere inviati al momento opportuno.

Nel caso lo studente non riesca ad implementare l'attesa asincrona, potrà implementare una gestione dei messaggi sequenziale, in cui ogni sender e receiver può gestire un solo messaggio alla volta e gli altri messaggi rimangono in attesa del loro turno. Questo influirà sulla valutazione del progetto ma la soluzione potrà essere comunque discussa e motivata durante l'esame orale.

Allo scadere del proprio tempo di attesa ogni messaggio può
i) venire inviato al receiver (tramite Q o SH in base al campo Type del messaggio) se si trova nel processo sender corretto (rispetto al campo IDSender del messaggio),
oppure
ii) venire inviato al prossimo processo sender S2 tramite la relativa PIPE1/FIFO1 che collega S1 ad S2.

La modalità di lettura dei messaggi da file F0.csv è a vostra discrezione. Potete leggere tutti i messaggi e mantenerle in memoria (array di struct message), oppure leggere i messaggi da file uno ad uno.

Questa procedura avviene all'interno di ciascun sender e receiver in base allo schema di Figura 1.

Al momento dell'invio del messaggio (sia nel primo che nel secondo caso) il processo aggiunge una riga al proprio file di log (tali file sono chiamati **F1.csv**, **F2.csv**, **F3.csv**, per i processi sender e **F4.csv**, **F5.csv**, **F6.csv** per i receiver, come mostrato in Figura 1) inserendo in tale riga i seguenti campi:

ID | *Message* | *IDSender* | *IDReceiver* | *TimeArrival* | *TimeDeparture*

dove

- TimeArrival: è l'orario di arrivo del messaggio nel processo (se S2, S3, R1, R2, o R3), oppure lettura dal file F0.csv se sono il processo S1;
- TimeDeparture: è l'orario di partenza del messaggio dal processo (Consiglio: usare il tempo immediatamente precedente/successivo alla scrittura su PIPE, FIFO, Q, o SH);

Nel caso in cui il messaggio non debba essere inviato dal sender S1, esso viene quindi passato a S2 utilizzando PIPE1/FIFO1, e se non deve essere inviato nemmeno da questo viene poi passato a S3 utilizzando PIPE2. In ogni sender il messaggio deve attendere il relativo tempo di attesa (DelS1, DelS2, DelS3 rispettivamente) prima di essere inviato ad un receiver o al prossimo sender.

Quando il messaggio viene ricevuto da un receiver segue poi il percorso delle pipe PIPE3/FIFO3 e/o PIPE4/FIFO4 (in base al receiver in cui arriva) per raggiungere il receiver R1. Quindi tutti i messaggi devono alla fine raggiungere il receiver R1. Si noti che nel receiver R3 i messaggi dovranno attendere ancora DelS3 secondi, nel receiver R2 dovranno attendere DelS2 secondi e nel receiver DelS1 dovranno attendere DelS1 secondi. Anche i receiver tengono traccia dei messaggi che sono transitati da essi, con relative informazioni e tempistiche (come specificato sopra per il file F1.csv) nei relativi file F4.csv, F5.csv e F6.csv.

Esempio di comunicazione del messaggio con id=1 in Figura 1 File F0.csv:

- viene prima caricato dal file F0.csv in S1, dove attende 2 secondi
- viene poi passato da S1 a S2 tramite PIPE1/FIFO1, e S1 tiene traccia della comunicazione scrivendo in F1.csv
- il messaggio attende 5 secondi in S2
- viene poi passato da S2 a S3 tramite PIPE3/FIFO3, e S2 tiene traccia della comunicazione scrivendo in F2.csv
- il messaggio attende 3 secondi in S3
- viene poi passato da S3 a R2 tramite message queue Q, e S3 tiene traccia della comunicazione scrivendo in F3.csv
- il messaggio attende 5 secondi in R2
- viene poi passato da R2 a R1 tramite PIPE4/FIFO4, e R2 tiene traccia della comunicazione scrivendo in F5.csv

- il messaggio attende 2 secondi in R1
- viene poi eliminato da R1 ed R1 tiene traccia del messaggio scrivendo in F6.csv

La **comunicazione tra senders e receivers** può avvenire in tre diverse modalità (coda di messaggi Q, memoria condivisa SH o fifo FIFO) che devono essere opportunamente implementate dallo studente per assicurare che i messaggi seguano il corretto percorso nei tempi specificati nel file F0.csv. I meccanismi di attesa nei processi sender e receiver possono essere implementati con modalità scelte dallo studente in modo che le specifiche siano soddisfatte.

Il processo disturbatore (denominato **Hackler** in Figura 1) carica dal file F7.csv (un esempio di tale file è allegato al template del codice) una lista di azioni di disturbo nella forma

ID | *Delay* | *Target* | *Action*

dove:

- Id: è l'identificativo sequenziale dell'azione di disturbo (e.g., 1, 2, …, n)
- Delay: è il ritardo con cui tale azione deve essere eseguita (e.g., 3 sec)
- Target: è l'id del processo (sender o receiver) a cui l'azione è rivolta (e.g., S1, S2, S3, R1, R2, o R3)
- Action: è uno dei tre valori "IncreaseDelay", "RemoveMsg" o "SendMsg", "ShutDown".
  - l'azione "IncreaseDelay" aumenta di 5 secondi il delay di tutti i messaggi in attesa nel processo target
  - l'azione "RemoveMsg" rimuove tutti i messaggi in attesa nel processo target
  - l'azione "SendMsg" fa scadere immediatamente il tempo d'attesa di tutti i messaggi in attesa nel processo target, quindi tali messaggi vengono inviati immediatamente al prossimo sender o al receiver (in base alle specifiche del messaggio stesso che non sono modificate),
  - l'azione "ShutDown" invia un segnale SIGTERM a tutti i sender ed i receiver i quali devono liberare le proprie risorse e terminare, consentendo la terminazione anche dei loro processi padre. *Dopo aver inviato questo segnale anche il processo Hackler termina. Si noti che l'azione ShutDown può essere inviata solo a tutti i processi, quindi il campo Target non viene considerato.*

Tali azioni devono essere eseguite inviando opportuni segnali ai processi target al momento corretto (indicato dal delay dell'azione). I PID dei processi sender e receiver su cui compiere l'azione devono essere reperiti dai file F8.csv e F9.csv. Tali file contengono un mapping tra l'id del sender ("S1", "S2" ed "S3") ed i relativi PID, e tra gli id dei receiver ("R1", "R2" ed "R3") ed i relativi PID.
In base all'azione indicata nel campo Action diversi segnali (a scelta dello studente) devono essere inviati. I processi sender e receiver devono essere in grado di ricevere e gestire tali segnali in modo opportuno.

Tutti i processi devono rimanere attivi fino a quando il processo Hackler invia ai processi sender e receiver il segnale SIGTERM (vedi azione "ShutDown"). I processi SenderManager e ReceiverManager devono attendere la terminazione dei propri figli per terminare. In fase di terminazione tutti gli strumenti di comunicazione tra processi (Coda Messaggi, Memoria Condivisa, Semafori, FIFO, PIPE etc.) devono essere opportunamente rimossi dai processi che li hanno generati. Ogni segnale non strettamente necessario per l'esecuzione del programma deve essere bloccato.

Infine, i processi SM, RM, ed Hackler salvano all'interno dei file F10.csv per ogni oggetto di Inter Process Communication (IPC) (i.e., PIPE1/FIFO1,….., PIPE4/FIFO4, FIFO, SH, Q etc.) il loro storico nel seguente formato (Nota. il nome del IPC deve essere scritto in MAIUSCOLO):

IPC | IDKey | Creator | CreationTime | DestructionTime

Attenzione: lo studente deve definire, utilizzando gli strumenti visti durante il corso, il meccanismo di condivisione di questo file al fine di evitare accessi in scrittura contemporanea. (Consiglio: usare un Semaforo del set di semafori SM)

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 *ProgettoSO2020-21/sistemi_operativi/system_call/***:

./sender_manager InputFiles/F0.csv & ./receiver_manager & ./hackler InputFiles/F7.csv
---

Gli **input** sono i file F0.csv ed F7.csv i cui nomi sono passati in argv all'avvio dei relativi programmi (come definito sopra)

Gli **output** sono i file F1.csv, F2.csv, F3.csv, F4.csv, F5.csv, F6.csv, F8.csv, F9.csv ed F10.csv. Tali file devono rimanere sul file system (directory OutputFiles/) anche dopo il termine dell'esecuzione per permettere una loro analisi successiva.

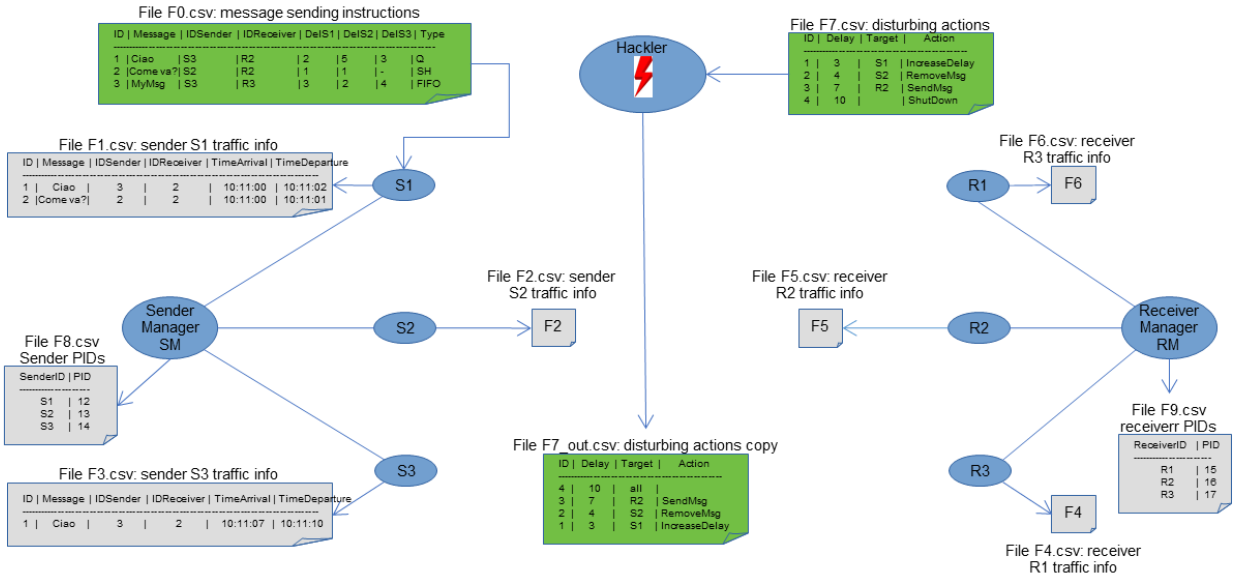
IMPORTANTE: I file di input e output devono rispettare i formati definiti sopra e riportati nei file d'esempio allegati utilizzando come delimitatore il carattere ';'.
IMPORTANTE: Gli header (prima riga delle tabelle) di ogni file devono corrispondere a quelli mostrati in Figura 1. Indipendentemente dal template che vi è stato fornito nel primo semestre, che riportava in alcuni casi header leggermente diversi.

Identificativi delle strutture di IPC
Usare le seguenti chiave per la generazione dei IPC.

IPC Keys	
Semaphore	**01110011**
Messagge Queue	**01110001**
Shared Memory	**01101101**
FIFO	**OutputFiles/my_fifo.txt**
FIFO1	**OutputFiles/my_fifo1.txt**
FIFO2	**OutputFiles/my_fifo2.txt**
FIFO3	**OutputFiles/my_fifo3.txt**
FIFO4	**OutputFiles/my_fifo4.txt**

Consegna del primo semestre

La consegna del primo semestre consente di acquisire il bonus di 3 punti. Tale consegna consente allo studente di avere anche un primo feedback sulla corretta compilazione, ed esecuzione del progetto (anche se verrà implementata solo una piccola parte), acquisizione degli input e generazione degli output (Figura seguente).



Il codice consegnato dovrà avere le seguenti funzionalità:

- esecuzione di SenderManager,
- generazione dei processi sender S1, S2, S3 come figli di SenderManager,
- generazione del file F8.csv da parte di SenderManager,
- lettura del file F0.csv da parte del sender S1, generazione delle strutture relative ai messaggi e memorizzazione in opportune strutture dati interne al processo,
- scrittura (senza attesa) del file F1.csv per tutti i messaggi letti da F0.csv (la comunicazione dei messaggi non può essere fatta perché richiede conoscenze che verranno fornite nel secondo semestre). Dopo tale scrittura i messaggi vengono eliminati, il processo S1 resta vivo per 1 secondo e poi termina,
- scrittura (senza attesa) dei file F2.csv e F3.csv con la sola intestazione da parte dei processi S2 ed S3, rispettivamente. Dopo tale scrittura i processi S2 ed S3 restano vivi rispettivamente per 2 e 3 secondi e poi terminano,
- terminazione di SenderManager quando tutti i suoi figli hanno terminato,
- esecuzione di ReceiverManager
- generazione dei processi receiver R1, R2, R3 come figli di ReceiverManager
- generazione del file F9.csv da parte di ReceiverManager
- scrittura (senza attesa) dei file F4.csv, F5.csv e F6.csv con la sola intestazione da parte dei processi R3, R2 ed R1, rispettivamente. Dopo tale scrittura i processi R1, R2 ed R3 restano vivi rispettivamente per 1, 2 e 3 secondi e poi terminano,
- terminazione di ReceiverManager quando tutti i suoi figli hanno terminato,
- esecuzione di Hackler
- lettura da parte del processo Hackler del file F7.csv. Dopo tale lettura il processo Hackler rimane vivo per 2 secondi e poi termina
- salvataggio delle varie azioni in un vettore di opportune strutture
- scrittura del contenuto delle strutture in un file F7_out.csv con lo stesso formato di F7.csv in ordine inverso (se tutto va a buon fine il file F7_out.csv dovrebbe essere uguale al file F7.csv

in ordine inverso). Nota bene: il file F7_out.csv non è richiesto nella consegna finale del progetto ma solo in quella intermedia (primo semestre), per questo non è rappresentato in Figura 1.

Non dovranno essere implementate:

- PIPE, FIFO, coda di messaggi, memoria condivisa e relativi meccanismi di comunicazione IPC
- meccanismi di attesa prima di inviare i messggi
- comunicazione dei messaggi tra sender e receiver (i.e., tutti i messaggi si fermeranno nel processo S1)
- invio di segnali da Hackler a sender e receiver

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.

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

1. La prima consegna dell'elaborato (primo semestre) consente di ottenere un punteggio massimo di tre trentesimi (3/30).
2. La seconda consegna dell'elaborato permette un punteggio massimo di ventiquattro24 trentesimi (24/30).
3. La consegna dell'esercitazione su MentOS/process-management, e del presente elaborato permette un punteggio massimo di ventisette trentesimi (27/30).
4. La consegna dell'esercitazione su MentOS/process-management, MentOS/deadlock-management, e del presente elaborato permette un punteggio massimo di ventinove trentesimi (29/30).

Il punteggio della prima consegna viene sommato ai punteggi della seconda consegna. La votazione massima è quindi 29+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".

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/scheduler_algorithm.c (da consegnare nel primo semestre)
/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. La cartella OutputFiles/ deve essere vuota e non ci devono essere eseguibili o file oggetto all'interno dell'archivio finale.

Nota bene: 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.