NOTES

Consigli Bini

- Realizzare solo i processi senza nè loro interazioni, nè logica di funzionamento.
  - Solo "scatole vuote" con fork(), execve(), e wait()
- Aggiungere una comunicazione "dummy" fra processi ancora vuoti: pipes, messaggi, semafori, segnali, etc.
- Aggiungere la logica di funzionamento
- (forse però, come suggerisce Gunetti, è meglio testare con i segnali per ultima cosa)

Consigli Gunetti

- Sviluppo Incrementale
- Incomincia con parametri bassi (ma sempre ben proporzionati!)
  - Può aiutare a capire quando (e perchè) vi siano problemi
  - ex. simula il viaggio di un singolo taxi, poi aggiungine qualcuno per vedere come reagiscono al conflitto
- Mantieni semplice la generazione delle richieste
  - Non è il punto principale, è più importante gestire i taxi !
- Inserire la gestione dei segnali dopo il codice di base
  - "Scardina il flusso di esecuzione"
  - Possono essere ricevuti da processi anche quando questi sono addormentati su sem o msgq : come reagisce ? La wait è finita con successo o perchè sono uscito dall'handler ?

SINCRONIZZAZIONE DI INIZIO SIMULAZIONE

1. 2 semafori, SEM_KIDS e SEM_START
2. master setta SEM_KIDS a 0 e SEM_START a 1
3. master si inizializza, poi fork delle sorgenti e semop di -SO_SOURCES ("semop-wait") su SEM_KIDS
4. ogni sorgente si inizializza, fa "semop-signal" (+1) su SEM_KIDS e poi wait for zero (semop-wait di 0) su SEM_START (aspetta l'inizio simulazione)
5. quando SO_SOURCES processi hanno fatto +1 su SEM_KIDS, master si sblocca
6. master setta SEM_KIDS a 0 e fa lo stesso procedimento dei punti 3. - 5. per creare i taxi (con una semop-wait di -SO_TAXI)
7. master setta SEM_KIDS a 0 e fa un procedimento simile ai punti 3. - 5. per creare il processo printer (con una semop-wait di -1)
8. master richiede di digitare ENTER (wait for input) per iniziare o Ctrl-C per terminare (così il prof può vedere i pid delle sorgenti)
9. ENTER : master fa partire la simulazione con una semop-wait di -1 su SEM_START, poi parte alarm(SO_DURATION)

Note sulla traccia del progetto (ver. "DEFINITIVA" 17/12/2020)

1.1 Mappa della città

- Array unidimensionale di Celle in shared memory (con macro di indicizzazione)
- 8 celle adiacenti accessibili, flag per indicarle un Hole
    - Numero massimo di Holes possibili ? (in proporzione alla griglia)

1.2 Richieste

- SO_SOURCES processi "sorgente", flag SRC_CELL per indicare una cella sorgente
    - Passo tramite execve info per accedere alla propria cella ?
    - Sarebbe in sola lettura, forse nemmeno necessario
        - Nel caso di una msgq (coda di messaggi) per ogni sorgente : dovrebbe accedervi per scrivere l'id della sua coda
- *Unica msgq per le richieste ? Oppure una per ogni sorgente ?*
    - Cambia anche il criterio di movimento dei taxi !
    - Una sola (condivisa)
        - - "Poco" spazio per le richieste (di solito 4kB)
            - Con min 8 byte per richiesta (forse ne servono più), ci stanno max 512 richieste in contemporanea
                - - Una sorgente rischia di non riuscire più a caricarne nella msgq
                - - Se ad inizio simulazione ogni sorgente mettesse 1 richiesta, con SO_SOURCES > 512 si andrebbe in deadlock.
                - Potrebbe essere un problema se i prof decidessero di modificarlo.
        - + Facilita la gestione delle risorse IPC (una sola msgq a cui pensare)
            - Semplifica anche il codice ?
    - Una per ogni cella sorgente
        - - Rischio di spreco di memoria
            - Configurazione "dense" : 190 sorgenti * 4kB = 760kB di msgqs
        - + Nessun rischio di starvation (attesa sulla coda) per le sorgenti
            - Vi saranno sempre abbastanza richieste per i taxi
            - Nella configurazione "large" questo è fondamentale
        - Complica davvero il codice ?
            - Il master non ha bisogno di accedervi (potrebbe avere msgq / pipe per altro)
            - Una sorgente deve preoccuparsi solamente della propria msgq (e cancellarla a fine simulazione / terminazione precoce)
            - Un taxi vede solo la msgq della cella (sorgente) che ha raggiunto
                - Ex. la sorgente scrive il proprio msgq_id nella cella
- *Sincronizzazione iniziale ?*

"Tutti i taxi partono insieme solo quando ci sono almeno NUM REQUEST richieste nella coda"

- Guardare esempio in test-sem-cook.c, lezione semafori (sync via due semafori)
- Flusso iniziale semplificato :
    - master crea griglia, assegna celle Hole, Src, inizializza array di semafori e msgqs (le sue)
    - master fa fork delle sorgenti (passa loro la posizione sulla griglia), che inizializzano il loro e vanno in wait su un semaforo
        - (Creano e) caricano qualche richiesta, inizializzano le proprie strutture dati / variabili
    -

Quando tutte le sorgenti sono pronte, master NON le sblocca, ma fa fork dei taxi, che si preparano e vanno anche in wait
        - I taxi, tra tutte le cose, devono anche leggere la griglia e salvare le posizioni delle sorgenti
    - Tutti i processi son pronti --> parte la simulazione, master li sblocca e alarm(SO_DURATION)
- *Pattern di generazione richieste ?*

"Richiesta periodica oppure burst ogni tot"

- Meglio burst ogni secondo
    - Caso msgq singola ?
        - Potrebbe non riuscire ad inviarle tutte
        - IPC_NOWAIT ?
            - Se ha altro da fare sì, ma una sorgente non dovrebbe fare altro a parte mandare richieste, può rimanere in wait
            - Accesso atomico in scrittura garantito a priori per le msgq
- *Richiesta da terminale ?*
    - Invio di segnale via kill da terminale (SIGINT o SIGUSR1 ?) al processo sorgente (top per vederne il pid) :
        - Creo taxi e sorgenti ma aspetto Enter da tastiera per far partire tutto ?
        - Per dare tempo al prof di leggersi i pid delle sorgenti
    - Se la coda è piena ?
        - Come per i burst di richieste, in teoria posso aspettare
        - Dovesse scattare l'alarm mentre è in wait per la richiesta del prof (codice dell'handler), posso ricevere l'alarm quando ho finito con questa
    - Se uso SIGINT per la terminazione precoce, lo maschero nell'handler ?

1.3 Movimento dei Taxi (situazione di stallo)

- "Se un taxi non si muove..."
    - Definire "movimento" : spostamento da una cella ad un'altra completato (accesso al semaforo della cella riuscito)
    - Su successo, devo fare ripartire il timer (alarm(SO_TIMEOUT))
    - La prima volta lo faccio partire ad inizio simulazione (appena sbloccato dal master)

(Da 1.2)
"I processi taxi prelevano le richieste, e le richieste possono essere servite soltanto da un taxi che si trova nella cella di origine della richiesta"

- Un taxi deve :
    - Prelevare una richiesta (con criterio di convenienza)
    - Spostarsi sulla src cell della richiesta
    - Spostarsi sulla dest cell della richiesta
- L'ordine di queste operazioni dipende soprattutto dalla scelta tra una o più msgq per le richieste

"I taxi se sono vicino alla richiesta la prendono"

- *Convenienza di una richiesta*
    -

Shortest path in base al time to sleep ?
        - Calcolo complicato, quasi sicuramente dura più di SO_TIMEOUT
    - Manhattan Distance per ogni sorgente
    - SE ho una singola msgq (non devo andare su src cell della richiesta per prelevarla)
        - Prima di muovermi ordino l'array con le posizioni delle sorgenti in base alla MD (O(nlogn))
            - Devo scrivermi la funzione di sorting...
        - Cerco sulla coda un msg con mtype uguale alla posizione più vicina
        - Se non lo trovo (IPC_NOWAIT) cerco la seconda più vicina e così via
        - Una volta trovata inizio a spostarmi verso src cell, e una volta raggiunta mi sposto su dest cell
    - ELSE (una msgq per sorgente)
        - Per sapere se per una cella sorgente ci sono richieste in coda, devo andarci !
        - Uso la MD per calcolare solo la posizione più vicina (O(n))
        - Mi sposto su src cell, leggo l'id della msgq
        - Se c'è una richiesta, la ritiro e vado a dest cell
        - Altrimenti calcolo di nuovo la src cell con min MD e ripeto !
    - In generale, considerando che il timer di un taxi viene resettato ogni volta che si sposta, conviene che giri a vuoto piuttosto che vada in wait sulle msgq
    - Potrebbero comunque morire (probabile) se circondati da celle piene di taxi o inaccessibili !
- Nel proc. taxi, mascherare l'alarm in sezione critica (?)
    - L'alarm indica che il taxi deve morire, non dovrebbe essere necessario
    - Alcune variabili devono essere accessibili anche nell'handler
- *Terminazione di un taxi*
    - "Per cause naturali" a fine simulazione (ex. ricevuto SIGINT)
    - Timeout
        - Il master deve distinguere il tipo di terminazione
        - Serve distinguere se un taxi sia stato creato prima o durante la simulazione ?
            - ex. con un valore flag passato come parametro ?
    - Il taxi deve mantenere / calcolare delle statistiche da inviare al master subito prima di terminare (vedi 1.5)
- *"Respawn" taxi ?*
    - Da comunicare con l'exit status ?
    - Poi dal while(wait) il master ne forka un altro
    - Legge dalla coda / pipe delle statistiche le informazioni del figlio appena terminato
- *Per le statistiche dei taxi ?*
    - msgq comune a master e taxis
    - pipe tra master (read) e taxis (write), ma non per le sorgenti

1.4 Inizio e terminazione

- Gestire se il timer venga annullato da altri segnali (man 3 alarm)

1.5 Stampa

- *Stampa ogni secondo*
    -

Lettura veloce del master sulla griglia ogni alarm(1) ?
        - Vi fosse un context switch durante la stampa verrebbe fuori uno schifo
        - Bisogna sincronizzare con un semaforo di sicuro i taxi **(anche le sorgenti ?)**
        - Li faccio sincronizzare nel codice o tramite segnale (e poi semop nell'handler) ?
    - Se l'alarm è usato per la simulazione, come ne faccio scattare uno ulteriore ?
        - fork di un processo che si occupa della stampa
        - alarm di solo 1 secondo nel master ripetuto per SO_DURATION volte
        - Uso funzioni get / set itimer
- *Raccolta delle statistiche ?*
    - Devono arrivare al master prima o poi
    - **numero viaggi** (non serve sapere quali: le richieste non vanno ricordate una volta prelevate)
        - Ne tengo il conto...
        - successo: ...in ogni taxi, nella struct TaxiStats (inviata poi al master).
        - abortiti: ...nel master, tramite la terminazione per timer di un taxi.
        - inevasi ("pendenti, ancora in coda") :
            - ...in ogni sorgente : a fine simulazione invia al master il numero di messaggi ancora nella coda (tramite exit status o pipe o msgq)
            - ...nel master se la coda è una sola (e quindi l'ha creata questo)
    - TOP_CELLS : ogni cella tiene il conto di quanti taxi la hanno attraversata, "classifica" stilata a fine simulazione
        - Ok, servirebbe anche qui una funzione di sorting, tanto vale farla a sto punto
    - Ogni taxi mantiene una struct TaxiStats (e la invia al master prima di terminare)
        - numero di celle attraversate (in totale)
        - durata del singolo viaggio più lungo (somma dei tempi di nanosleep)
        - numero di richieste completate con successo (nella traccia dice "raccolte", nel caso sarebbero solo n completate più 1 abortita)
        - pid del taxi

"Se una cella a fine esecuzione è contemporanemente una SO TOP CELLS ed una SO SOURCES, come la evidenzio?(dato che rientra in entrambe le categorie e sarebbe (forse) utile poter distinguere le celle evidenziate perchè appartenenti ad una categoria o all'altra)."

- Secondo il prof il calcolo delle celle top deve escludere le sorgenti, che forse sarebbero sempre le più visitate.

Gli altri punti della traccia non necessitano precisazioni

Segnali (da pensarci bene più avanti)

- Da gestire
    - flusso nell'handler per ognuno di questi
    - quali segnali vadano (o meno) mascherati durante l'esecuzione dell'handler
- master

    - SIGALRM : la simulazione è conclusa
    - SIGINT : terminazione forzata (quindi anche per i figli)
    - Stampa ogni secondo ? (potrebbe servire un altro processo)
- sorgente
    - SIGALRM : burst di richieste
    - SIGINT : terminazione forzata
    - SIGUSR1 : richiesta da terminale
- taxi
    - in ogni caso invia stats al master
    - SIGALRM : timeout, exit(TIMEOUT_STATUS)
    - SIGINT : terminazione forzata

Strutture Dati

- Cella
    - tempo per attraversarla (per nanosleep)
    - capacità di attraversamento (per inizializzare il semaforo, serve ricordarla nella cella ?)
    - n taxi passati da qui
    - flags (sorgente? hole?)
- Richiesta (msg)
    - mtype (usabile come posizione della src_cell, o sia di src che di dest)
    - (eventualmente) dest_cell
- Statistiche Taxi
    - già indicato in 1.5

IPC Resources

- Griglia di SO_WIDTH * SO_HEIGHT celle in shared memory
- Code di messaggi (msgqs)
    - 1 tra master e taxis per invio statistiche
    - 1 comune a tutti oppure SO_SOURCES per ogni sorgente per l'invio delle richieste
    - altre?
- (Array di) Semafori (da rivedere bene, anche la teoria)
    - SO_WIDTH * SO_HEIGHT per ogni cella (init. SO_CAP)
    - 1+ per sync iniziale tra master, sorgenti e taxis
    - 1 per sync ogni secondo (stampa)
        - stabilire bene quando e dove (nel codice) avviene questa sync