ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ - ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Λειτουργικά Συστήματα — Άσκηση 3

Εργασία Φοιτητή:
Δαμιανός Ντούμη - Σιγάλας, 6157
nsigalas@ceid.upatras.gr

Multi-process implementation

`multiproc_1.c` `multiproc_2.c`

Στα πρώτα δύο ερωτήματα της άσκησης καλούμαστε να υλοποιήσουμε τη ζητούμενη λειτουργικότητα εφαρμόζοντας μια πολυδιεργασιακή προσέγγιση. Αρχικά αφού γίνει ο κατάλληλος έλεγχος ότι ο χρήστης έχει εισάγει τουλάχιστον ένα αλφαριθμητικό, τότε δημιουργούνται τόσες διεργασίες όσα και τα string εισόδου (`πλήθος = argc-2`) κάθε μια από τις οποίες αναλαμβάνει να εκτυπώσει στην οθόνη rep φορές *μόνο* το string που της αντιστοιχεί κάνοντας χρήση της `display(char*)` που μας δίνεται. Σε αυτό το σημείο, χωρίς να έχει εφαρμοστεί κάποια τεχνική συγχρονισμού των διεργασιών, μια πιθανή έξοδος του προγράμματος είναι η επόμενη:

```
prompt $ ./multiproc_1 3 operating systems
[13914,139832146802432] o[13915,139832146802432] spyesrtaetmisn
[13915,139832146802432] sgys
[13914,139832146802432] otpeemrsa
[13915,139832146802432] styisntge
[13914,139832146802432] ompse prompt $
rating
```

Δύο προβλήματα γίνονται αμέσως αντιληπτά. Πρώτον, οι χαρακτήρες των string εκτυπώνονται με ανακατεμένη σειρά και δεύτερον, το prompt του bash εκτυπώνεται σε λάθος σημείο. Το τελευταίο συμβαίνει διότι η διεργασία-γονέας δεν περιμένει την ολοκλήρωση των παιδιών της, τερματίζοντας πριν από αυτά. Για την λύση του αρκεί να εξαναγκάσουμε τον γονέα να περιμένει όλες τις διεργασίες-παιδιά που έχει δημιουργήσει, με την χρήση της κλήσης συστήματος `wait` .

Το πρώτο πρόβλημα δημιουργείται καθώς μεταξύ της εκτύπωσης κάθε χαρακτήρα η συνάρτηση `display` προσθέτει τουλάχιστον 100 μικροδευτερόλεπτα αδράνειας στο thread που την εκτελεί, με τον scheduler του λειτουργικού συστήματος να αποφασίζει την αντικατάσταση της τρέχουσας διεργασίας με κάποια άλλη, η οποία θα συνεχίσει την λειτουργία της από το σημείο που έχει μείνει. Για την επίλυση του προβλήματος, πρέπει να επιτευχθεί ο αμοιβαίος αποκλεισμός (mutual exclusion) των διεργασιών όταν εκτελούν τον κώδικα της συνάρτησης `display` ώστε κάθε φορά μόνο μια διεργασία να γράφει ολοκληρωμένα το string που της αντιστοιχεί στην οθόνη και έπειτα να επιτρέπει σε κάποια άλλη να πάρει τη θέση της.

Για τον σκοπό αυτό, θα χρησιμοποιηθεί ένας σημαφόρος. Οι σημαφόροι αποτελούν μιά δομή δεδομένων που επιτρέπει των συγχρονισμό πολλαπλών διεργασιών. Για την υλοποίηση του αμοιβαίου αποκλεισμού απαιτείται ένας δυαδικός σημαφόρος που παίρνει τιμές στο σύνολο $\{0, 1\}$ και έχει αρχικοποιηθεί στην τιμή 1. Στην περίπτωση που κάποια χρονική στιγμή ο σημαφόρος έχει την τιμή 0, σημαίνει ότι κάποια άλλη διεργασία βρίσκεται στην κρίσιμη περιοχή και η διεργασία που προσπάθησε να μπει στην δική της κρίσιμη περιοχή πρέπει να περιμένει. Αντίθετα αν ο σημαφόρος έχει την τιμή 1, σημαίνει οτι καμία διεργασία δεν βρίσκεται σε κρίσιμη περιοχή και η τρέχουσα διεργασία μπορεί να προχωρήσει στην προσπέλαση της.

Για την υλοποίηση αυτής της λειτουργικότητας έχουν δημιουργηθεί οι εξής συναρτήσεις (οι οποίες χρησιμοποιούν το System V semaphore API):

1. `int create_semaphore(key_t key, int sem_flags, int init_value)`

   Δημιουργία του σημαφόρου με χρήση της `semget` και αρχικοποίηση του στην τιμή `init_value` με χρήση της `semctl`. Επιστρέφεται το id του δημιουργηθέντος σημαφόρου.

2. `int destroy_semaphore(int sem_id)`

   Διαγραφή του σημαφόρου με χρήση της `semctl`, για απελευθέρωση των πόρων του λειτουργικού συστήματος.

3. `int increment(int sem_id)`

   Αύξηση της τιμής του σημαφόρου κατα 1 με χρήση της `semop` ώστε κάποια από τις διεργασίες που βρίσκονται σε αναμονή (αν υπάρχουν) να ξεκινήσει την λειτουργία της.

4. `int decrement(int sem_id)`

   Μείωση της τιμής του σημαφόρου κατά 1 με χρήση της `semop` ώστε να μπλοκαριστεί η διεργασία σε περίπτωση που το αποτέλεσμα είναι αρνητικό (δηλαδή η τιμή του σημαφόρου ήταν 0).

Με την χρήση των παραπάνω δημιουργείται ένας σημαφόρος αρχικοποιημένος στην τιμή 1. Πριν την έναρξη της κρίσιμης περιοχής του προγράμματός μας, δηλαδή την κλήση της `display` ή και της `init` στην περίπτωση του `multiproc_2.c` καλείται η `decrement` ώστε αν η τιμή του σημαφόρου ήταν 1 να γίνει 0 και να επιτραπεί η εκτέλεση του κρίσιμου κώδικα ενώ αν ήταν 0, να μπλοκαριστεί η διαδικασία μπαίνοντας σε αναμονή.

Επιπλέον στην περίπτωση του δεύτερου ερωτήματος, θα πρέπει όλες οι διεργασίες να έχουν ολοκληρώσει την κληση της `init` και μετά να συνεχίσουν την λειτουργία τους. Για να επιτευχθεί αυτό χρησιμοποιείται ένας σημαφόρος ο οποίος θα επιτρέψει την συνέχιση της ροής κάθε διεργασίας μόνο όταν έχει μετρήσει τόσες κλήσεις της `init` όσες και το πλήθος των εκτελούμενων διεργασιών.

Πλέον μετά την εφαρμογή αυτών των τεχνικών μια πιθανή έξοδος του προγράμματος είναι η παρακάτω:

```
prompt $ ./multiproc_2 3 operating systems
STARTING: pid 28200, tid 140528943884032
STARTING: pid 28201, tid 140528943884032
[28200,140528943884032] operating
[28201,140528943884032] systems
[28200,140528943884032] operating
[28201,140528943884032] systems
[28200,140528943884032] operating
[28201,140528943884032] systems
prompt $
```

Multi-thread implementation

`multithread_1.c` `multithread_2.c`

Σε αυτά τα δύο προγράμματα αντί να δημιουργούνται τόσες διεργασίες όσα και τα string που δίνει ο χρήστης ως ορίσματα κάθε μια εκ των οποίων αναλαμβάνει να εκτυπώσει rep φορές το string που της αντιστοιχεί, θα πρέπει να δημιουργείται ο αντίστοιχος αριθμός νημάτων (threads) της ίδιας διεργασίας όπου το καθένα θα επιτελεί την ίδια εργασία.

Αφού κάθε νήμα πρέπει να εκτυπώνει rep φορές το string που του αντιστοιχεί για τον σκοπό αυτό δημιουργείται η συνάρτηση `void* thread_runner(void *p)` την οποία καλεί κάθε νήμα κατά την δημιουργία του. Παρακάτω παρατίθεται η έξοδος για μία εκτέλεση του προγράμματος `multithread_2.c` χωρίς να έχει εφαρμοσθεί κάποια τεχνική για τον συγχρονισμό των νημάτων.

```
prompt $ ./multithread_2 3 operating systems
STARTING: pid 25839, tid 140057900754688
[25839,140057900754688] oSTARTING: pid 25839, tid 140057892361984
[25839,140057892361984] spyesrtaetmisn
[25839,140057892361984] sgy
[25839,140057900754688] ospteermast
[25839,140057892361984] siynsgt
[25839,140057900754688] oepmesr
ating
prompt $
```

Παρατηρούμε το αντίστοιχο πρόβλημα της εκτύπωσης χαρακτήρων από πάνω από ένα νήμα ταυτόχρονα. Αντίστοιχα με την λογική που αναπτύχθηκε παραπάνω, πρέπει να πετύχουμε τον αμοιβαίο αποκλεισμό των νημάτων έτσι ώστε μόνο ένα νήμα κάθε χρονική στιγμή να εκτελεί τις κρίσιμες περιοχές κωδικά του, δηλαδή στην περίπτωση μας τις συναρτήσεις `init` και `display`.

Η βιβλιοθήκη `pthread` μας παρέχει τα mutexes (`pthread_mutex_t`), ένα εργαλείο με λειτουργία παρόμοια αυτής του δυαδικού σεμαφόρου. Το mutex λειτουργεί σαν μια κλειδαριά με την διαφορά ότι μόνο ένα thread κάθε χρονική στιγμή μπορεί να το κλειδώσει (lock) και κανένα άλλο νήμα δεν μπορεί να το χρησιμοποιήσει μέχρι ο κάτοχος του να το ξεκλειδώσει (unlock). Εξασφαλίζοντας ότι κατά την είσοδο του σε κρίσιμη περιοχή το νήμα δεσμεύει το mutex επιτυγχάνουμε τον αμοιβαίο αποκλεισμό καθώς κανένα άλλο νήμα δεν μπορεί να το δεσμεύσει για την εκτέλεση της δικής του κρίσιμης περιοχής μέχρι το αρχικό να το αποδεσμεύσει.

Επιπλέον επειδή πρέπει να ολοκληρωθούν όλες οι κλήσεις της `init` πριν ξεκινήσουν οι κλήσεις της `display` από οποιοδήποτε νήμα, ανάμεσα στις δύο κρίσιμες περιοχές πρέπει να εισαχθεί ένα barrier (`pthread_barrier_t`), το οποίο εξασφαλίζει ότι όλα τα threads θα έχουν ολοκληρώσει μέχρι αυτό το σημείο, ώστε να συνεχίσουν έπειτα την λειτουργία τους. Επομένως η έξοδος που προκύπτει από μία εκτέλεση του `multithread_2.c` είναι η ακόλουθη:

```
prompt $ ./multithread_2 3 operating systems
STARTING: pid 25675, tid 140437636089600
STARTING: pid 25675, tid 140437627696896
[25675,140437627696896] systems
[25675,140437627696896] systems
[25675,140437627696896] systems
[25675,140437636089600] operating
[25675,140437636089600] operating
[25675,140437636089600] operating
prompt $
```