Readme

Λαμπρόπουλος Κωνσταντίνος
ΑΜ : 1115201800092

Περιεχόμενα

1 Εισαγωγή

Στην εργασία αυτή ζητήθηκε να υλοποιηθεί ένα πρόγραμμα με σκοπό τον συγχρονισμό πολλών processes (IPC).Η λειτουργία το παρόντος προγράμματος είναι η επικοινωνία της μητρικής διεργασίας (η οποία δημιουργείται όταν κάνουμε execute το πρόγραμμα) μ'ένα σύνολο παιδιών (ο αριθμός των οποίων δίνεται σαν όρισμα από την γραμμή εντολών). Ειδικότερα ένα παράδειγμα εκτέλεσης του προγράμματος είναι το εξής :
.\main -p PROCESS_VAL -t TEXT_FILE -r REQUEST_VAL -g GRAGING,
όπου : PROCESS_VAL ο αριθμός των διεργασιών,TEXT_FILE το όνομα του κειμένου που θα διαβαστεί,REQUEST_VAL ο αριθμός των αιτημάτων που θα διεκπαιρεώσει η κάθε διεργασία και GRADING ο βαθμός κατάτμησης (από πόσες γραμμές θα αποτελούνται τα segments που θα χωριστεί το αρχείο κειμένου).Άμα δωθούν λάθος τα ορίσματα θα υπάρχει undefined behaviour.Αν δεν δωθούν όλα τα ορίσματα θα υπάρχει τερματισμός του προγράμματος.
Η λογική του προγράμματος είναι ότι το parent process παίρνει τα ορίσματα από την γραμμή εντολών,δημιουργεί ένα logfile για τους χρόνους διεκπαιρέωσης των αιτημάτων, διαγράφει αν υπάρχουν παλιά logfiles (και δικά του και των υπόλοιπων processes,τα οποία αποθηκεύονται στον φάκελο logfiles που δημιουργείται μόνος του στο current directory), και δημιουργεί ένα αρχείο που λέγεται temp.Το αρχείο temp είναι ο τρόπος αποστολής και παραλαβής αντίστοιχα των αιτημάτων των παιδιών από των πατέρα,ενώ όλη η υπόλοιπη επικοινωνία γίνεται μέσω pipes.Η επιλογή αυτή έγινε γιατί αν δίαβαζα τα αιτήματα των παιδιών μέσω pipes υπήρχε το θέμα ότι η read() συνάρτηση μπλοκάρει μέχρι να λάβει input και επίσης χάνεται η λογική του FIFO, γιατί πάντα το process[0] θα γράφει πρώτο.Το temp file διαγράφεται αυτόματα με τον τερματισμό του προγράμματος.Το πρόγραμμα τερματίζει όταν ο πατέρας σταματήσει να λαμβάνει input από τα παιδιά και οπότε περιμένει να τερματίσουν ,καθαρίζει τον χώρο (destroy shared memory,semaphores) και τερματίζει το πρόγραμμα.Στα επόμενα sections αναλύονται και οι βασικές συναρτήσεις και στο τέλος υπάρχει το compilation section.

2 Functions

2.1 Helpful Parent Functions

```
void CreateDirectory();
```

Αρχικοποιεί το directory logfiles για να γράψουν τα logs εκεί. Καλεί μέσα της την DeleteDirectory η οποία ελέγχει αν υπάρχει ήδη το directory και αν ναι,το διαγράφει με ότι έχει μέσα.

```
int DeleteDirectory(const char* dirname);
```

Ελέγχει αν το directory με path dirname υπάχει και έχει αρχεία μέσα ,αν όχι επιστρέφει 1 αλλιώς τα διαγράφει και σε περίπτωση επιτυχίας επιστρέφει 0.

```
void CreateParentLog();
```

Δημιουργεί το log της μητρικής διεργασίας και αν υπάρχει ήδη διαγράφει το υπάρχον και δημιουργεί ένα κενό logfile.

```
int CreateTemporaryFile();
```

Δημιουργεί το temp αρχείο στο οποίο θα γράφονται τα αιτήματα των διεργασίων (παιδιών) και επιστρέφει το file descriptor του.

```
    void CheckInput(int argc,char* argv[],int* Grade,int* Processes,int
    * Requests,char** file);
```

Δέχεται ως όρισμα τις μεταβλητές argc,argv[] ,ελέγχει αν είναι 9 ,αν όχι τερματίζει το πρόγραμμα.Αλλιώς ψάχνει τον πίνακα argv[] μέχρι το τέλος του και ελέγχει αν θα συναντήσει κάποιο από τα char* : -t,-p,-r,-g και μ'ένα σύνολο από booleans ενημερώνεται για να ξέρει ότι το επομένο iteration του argv[] πίνακα θα είναι η τιμή της αντίστοιχης μεταβλητής που θέλουμε.Επιστρέφει τις μεταβλητές Grade,Processes, Requests,file.

```
    void CheckPartitioning(int* partitions,int grade,char* filename);
```

Ανοίγει το αρχείο με όνομα filename και μέχρι να διαβάσει 0 bytes από την read() (δηλαδή EOF) επιστρέφει ένα char* ,το οποίο με την βοήθεια της strtok σπάμε με delimiter το "\n".Κάθε νέα γραμμή προσθέτουμε +1 στο counter μας.Τέλος βρίσκουμε πόσα partitions έχουμε (αφού γνωρίζουμε το σύνολο των γραμμών είναι μια απλή διαίρεση) και το επιστρέφουμε από τη μεταβλητή paritions.

```
    char* CreateSharedMemory(int* shmid,int grade);
```

Δημιουργεί το shared memory μας ,με χωρητικότητα MAX_BUFFER_SIZE*grade, και επιστρέφει το char* shm με το οποίο γράφουμε στο shared memory μαζί με το ID του shared memory.

```
    sem_t** CreateSemaphores(sem_t** Semaphore1,sem_t** Semaphore2,
    sem_t** Semaphore3,sem_t** Semaphore4,int Processes);
```

Δημιουργεί και αρχικοποιεί ένα σύνολο σημαφόρων που θα βρίσκονται σε shared memory (για να μπορούν να τα χρησιμοποιήσουν και τα άλλα processes) και τα επιστρέφει.Επίσης επιστρέφει έναν πίνακα από σημαφόρους ο οποίος έχει μέγεθος όσο τα processes.

```
    void Cleanup(sem_t* Semaphore1,sem_t* Semaphore2,sem_t* Semaphore3,
    sem_t* Semaphore4,sem_t** shmsem,char* shm,int shmid,int Processes)
    ;
```

Διαγράφει το αρχείο temp,τους σημαφόρους (ελευθερώνει και τον χώρο από το shared memory) όπως και το shared memory μέσα από το οποίο διαβάσαν τα requests τους τα child processes.

```
    void WriteTime(struct timespec start,struct timespec end,int
    partition);
```

Γράφει σ'ένα logfile (η μητρική διεργασία πάντα) τον αριθμό του segment που υπήρχε στο shared memory,όπως και τον χρόνο απ'όταν το ανέβασε μέχρι όταν το διέγραψε από το shared memory.

```
void FindPartition(int part,int partitions,int grade,char* filename
,char* shm,sem_t** shmsem,int Processes,struct timespec* start);
```

Ανοίγει το αρχείο με όνομα filename και κάνει πολλαπλές read().Σπάει το char* που επιστρέφει η read() με την βοήθεια της strtok() με delimiter ”\n” και έτσι μετράει κάθε γραμμή. Όταν φτάσει στην γραμμή (part - 1)*grade + 1 και μέχρι την γραμμή part*grade ,περνάει όλες τις γραμμές στο shared memory.Τέλος μόλις περασθούν οι γραμμές στο shared memory αρχίζει και μετράει τον χρόνο που θα αποθηκευτεί στο logfile και ενημερώνει όλα τα παιδιά μέσω του πίνακα σημαφόρων ότι μπορούν να διαβάσουν το segment από το shared memory.

2.2 Main Parent Function

```
void CreateProcesses(int argc,char* argv[]);
```

Αυτή αποτελεί την βασική συνάρτηση του αρχείου.Αρχικά δημιουργεί το Directory logfiles, παίρνει τα ορίσματα που έχουν εισαχθεί από την γραμμή εντολών ,σπάει το text file σε partitions, δημιουργεί τον πίνακα με τα pid_t των child processes και τα αντίστοιχα pipes.Αρχικοποιεί τα pipes, και τα semaphores.Επίσης μέσα σ’ένα for loop,όπου σε κάθε iteration δημιουργείται και ένα παιδί, κλείνει τα pipes που δεν χρειάζεται (τα pipes είναι δύο πίνακες του τύπου Array[ProcessesNumber][2]). Ειδικότερα για κάθε παιδί κάνουμε iterate και τους δύο πίνακες από pipes και κλείνουμε κάθε σε κάθε iteration που το $i \neq j$ (αφού το ith στοιχείο του πίνακα τον pid_t θέλουμε να χρησμιποιεί το ith pipe του πίνακα των pipes). Και μετά τα παιδιά περιμένουν να τα συνεχίσει η μητρική διεργασία.
Έπειτα η μητρική διεργασία στέλνει σε όλα τα παιδιά τις πληροφορίες του αρχείου (Partitioning,Grading,number of Requests) δημιουργεί το αρχείο temp όπως και το parent log και αρχικοποιεί το shared memory.Στέλνει signal σταματήσει παιδιά να συνεχίσουν την εκτέλεσή τους και περιμένει να λάβει μήνυμα ότι έχουν αρχικοποιήσει όλες τις μεταβλητές του προκειμένου να στείλουν αιτήματα.Όταν λάβει αυτό το μήνυμα μπαίνει σ’ένα while loop το οποίο τερματίζει όταν δεν υπάρχει άλλο input από τα child processes.
Στο while loop περιμένει να διαβάσει από το αρχείο μέσω ενός κοινού σημαφόρου (UseTempFile) ο οποίος είναι κοινός για όλες τις διεργασίες ώστε να εξασφαλισθεί ότι δεν θα διαβάσει η μητρική διεργασία την στιγμή που γράφει κάποιο παιδί στο αρχείο. Έπειτα στέλνει σε όλα τα παιδιά ποιό segment θα προστεθεί στο shared memory,αδειάζει τον χώρο του shared memory και αν δεν είμαστε στο πρώτο iteration του while loop γράφει και τον χρόνο που πήρε για να αδειάσει το shared memory. Τέλος βρίσκει το segment που ζητήθηκε και το προσθέτει στο shared memory και περιμένει να γίνει post το semaphore ChangeSharedMemory.Αυτό το semaphore βρίσκεται στην αρχή του while loop αλλά έχει αρχική τιμή 1 οπότε θα περάσει το sem_wait την 1η φορά και θα μπλοκάρει την 2η.
Τέλος οταν δεν λάβει άλλο input,περιμένει να τελειώσουν όλα τα παιδιά ,και καθαρίζει τον χώρο από τα semaphores και το shared memory.

2.3 Helpful Children Functions

```
int rand50();
```

Επιστέφει με πιθανότητα 50% 1 ,αλλιώς 0.

```
bool rand75();
```

Επιστέφει με πιθανότητα 75% true ,αλλιώς false.

```
void OpenSharedMemory(int* shmid,char** shm,int grade);
```

Ανοίγει το shared memory στο address space της(η κάθε διεργασία) και επιστρέφει το ID και ένα pointer στα δεδομένα του.

```
void CloseSharedMemory(char* shm);
```

Κάνει detach από το shared memory ώστε να μπορεί να διαγραφεί και να ελευθερωθεί ο χώρος στη μνήμη.

```
void WriteLog(char* shm,int line,int part,sem_t* semaphore,sem_t*
sem2,int* counter2,struct timespec start);
```

Ανοίγει ένα .log αρχείο με όνομα το pid του (από την getpid()) και ψάχνει στο shared memory την γραμμή που θέλει. Όταν την βρει βρίσκει τον χρόνο που χρειάστηκε για να διεκπαιρεωθεί το αίτημα και αποθηκεύει στο logfile τη γραμμή, τον αριθμό του segment και τον αριθμό της γραμμής ,και τον χρόνο που πήρε για να ολοκληρωθεί το αίτημα. Τέλος περιμένει 20ms και ελέγχει αν μπορεί να ενημερώσει το parent semaphore (αν δεν το κάνει αυτή τη στιγμή κάποια άλλη διεργασία), το ενημερώνει και επιστρέφει την συνάρτηση ReadFromFile ότι ολοκληρώθηκε 1 αίτημα.

2.4 Main Children Function

```
void ReadFromFile(int writer,int reader,sem_t* semaphore,sem_t*
shmsem,sem_t* semPar,sem_t* semChecker,sem_t* sem2);
```

Δίνει ένα τυχάιο seed με βάσει την getpid() στην srand(),γιατί αλλιώς όλες οι διεργασίες θα ζητήσουν τα ίδια segment και lines, διαβάζει από τον πατέρα τις πληροφορίες του κειμένου (Partitioning,Grading,number of Requests) ,ανοίγουν το αρχείο temp για να στέλνουν τα αιτήματά τους και κάνουν attach το shared memory.Μετά ακολουθεί ένα while loop,το οποίο τερματίζει όταν ολοκληρωθούν τα ζητούμενα αιτήματα.
Στο loop αυτό επιλέγεται με πιθανότητα 75% το ίδιο segment και line,αλλιώς ένα τυχαίο ,και περιμένει το process να γράψει στο temp file το αίτημά του. Στο temp file γράφουν ένα ένα με τη βοήθεια ενός σημαφόρου.Μόλις έχει σταλεί το αίτημα ξεκινάει να μετρά το χρόνος απόκρισης.Αν βρισκόμαστε και στο πρώτο iteration του while loop,ενημερώνουμε την μητρική διεργασία ότι μπορεί να αρχίσει να διαβάζει από το temp file.
Έπειτα υπάρχει ένα άλλο while loop στο οποίο τα παιδιά περιμένουν να ενημερώσει ο γονέας ότι υπάρχει νέο κομμάτι στο shared memory και διαβάζουν ποιό είναι αυτό.Αν αυτό είναι και αυτό που θέλει η διεργασία,σπάει από το loop και προχωράει να βρεί την γραμμή που θέλει.Αλλιώς περιμένει μέχρι να βρεθεί το segment που θέλει.

3 Compilation

Υπάρχει Makefile μαζί το οποίο με την εντολή make παράγει το εκτελέσιμο .\main. Ο τρόπος εκτέλεσης περιγράφεται και στην εισαγωγή αλλά ενδεικτικά ένα παράδειγμα είναι το εξής : .\main -t sometext.txt -p 10 -r 100 -g 10 .Αν το Makefile δεν λειτουργεί ένα από compilation command είναι το εξής : gcc -o main Parent.c Child.c Directory.c .