(ΕΡΓΑΣΙΑ 1)

ΚΟΛΟΚΥΘΑΣ ΧΡΗΣΤΟΣ

1115201700052

Περιεχόμενα

1. Εισαγωγή

Στην παρούσα αναφορά θα αναλυθεί πως αντιμετωπίστηκαν τα επιμέρους ερωτήματα που τέθηκαν στην εργασία 1 του μαθήματος «Λειτουργικά Συστήματα». Σε κάθε επιμέρους τμήμα περιγράφεται το ζητούμενο, παρατίθεται ο κώδικας που αναπτύχθηκε και αναφέρονται τυχόν δυσκολίες και πως αυτές αντιμετωπίστηκαν.

2. Έλεγχος παραμέτρων του προγράμματος

Σύμφωνα με την εκφώνηση, η διεργασία δέχεται ως όρισμα ένα αρχείο κειμένου, το πλήθος των διεργασιών παιδιών καθώς και τον αριθμό των δοσοληψιών στις οποίες εμπλέκεται καθεμία εκ των διεργασιών-παιδιών. Έχουμε λοιπόν 4 συνολικά παραμέτρους (πάντα το όνομα του προγράμματος αποτελεί την πρώτη παράμετρο στο πρόγραμμα). Ο έλεγχος για το πλήθος των παραμέτρων είναι απλός και φαίνεται παρακάτω

```
// Check if the user supplied the corrent number of parameters
if (argc != 4){
    // Let the user know the correct programma usage
    printf("Usage is %s <filename> <K children> <N transactions>\n", argv[0]);

    // Exit the program.
    exit(EXIT_FAILURE);
}

// Convert the arguments from string to int
k_child= atoi(argv[2]);
n_trans= atoi(argv[3]);
```

Εάν το πλήθος των παραμέτρων είναι διάφορο του 4, τότε το πρόγραμμα ενημερώνει τον χρήστη για την ορθή χρήση του και τερματίζει. Εάν το πλήθος των παραμέτρων είναι σωστό, τότε μετατρέπονται σε ακεραίους οι αριθμητικές παράμετροι (οι παράμετροι είναι όλοι strings αρχικά) και η εκτέλεση του προγράμματος συνεχίζει.

3. Ανάκτηση πληροφοριών για το αρχείο εισόδου

Η γονική διεργασία πρέπει να μετρήσει το πλήθος των γραμμών του αρχείου και να ενημερώσει τις θυγατρικές για αυτό.

Αρχικά, ανοίγουμε το αρχείο που δίνεται ως είσοδος. Αυτό γίνεται με την κλήση συστήματος (syscall) read. Η συνάρτηση αυτή επιστρέφει -1 εάν υπήρξε σφάλμα κατά το άνοιγμα του αρχείου. Ελέγχοντας την τιμή αυτή διακόπτεται η εκτέλεση του προγράμματος εάν το αρχείο δεν βρεθεί για παράδειγμα.

```c
// Open the file given as input.
input_fd = open(argv[1], O_RDONLY);

// Check for errors while opening the file
if (input_fd == -1){
    // Print the error associated with the errno set by open
    perror("Error: ");
    exit(EXIT_FAILURE);
}
```

Στο σημείο αυτό έχουμε δύο στόχους. Πρώτον, να μετρήσουμε το πλήθος των γραμμών στο αρχείο εισόδου και δεύτερον να αποθηκεύσουμε το αρχείο στην μνήμη του κυρίως προγράμματος σε τέτοια μορφή η οποία θα διευκολύνει την εξυπηρέτηση των αιτημάτων από τις θυγατρικές διεργασίες. Μια τέτοια δομή είναι ένας char**. Ο δείκτης αυτός δείχνει σε μια περιοχή με δείκτες σε χαρακτήρα. Καθένας από αυτούς τους δείκτες θα δείχνει σε ένα string που θα αντιστοιχεί μία γραμμή του αρχείου. Επιλέχθηκαν δείκτες, καθώς δεν είναι εκ των προτέρων γνωστά ούτε το πλήθος των γραμμών, ούτε το μήκος κάθε μίας.

Πρώτη μας κίνηση λοιπόν, είναι να υπολογίσουμε το μέγεθος του αρχείου. Αυτό γίνεται με χρήση της κλήσης συστήματος lseek. Με την συνάρτηση αυτή, μπορούμε να θέσουν το offset του αρχείου όπου θέλουμε. Η συνάρτηση επιστρέφει κατά πόσα bytes μετακινήθηκε αυτό το offset. Οπότε, εάν το θέσουμε στο τέλος του αρχείου με την επιλογή SEEK_END, τότε επιστρέφει το μέγεθος του αρχείου σε bytes. Τέλος, πρέπει να επιστρέψουμε το offset του αρχείου στην αρχή του για να συνεχίσουμε με την ανάγνωσή του. Τώρα, γνωρίζοντας το μέγεθος του αρχείου μπορούμε να δεσμεύσουμε μνήμη για αυτό.

```c
// Calculate the file size
file_size = lseek(input_fd, 0, SEEK_END);

// Check for errors while getting the file size
if (file_size == -1){
    // Print the error associated with the errno set by lseek
    perror("Error: ");
    exit(EXIT_FAILURE);
}

// Reset the file offset to the beginning
lseek(input_fd, 0, SEEK_SET);

// Allocate space for the entire file
file_buffer = malloc(file_size * sizeof(char));
```

Στη συνέχεια, με χρήση της κλήσης συστήματος read αντιγράφουμε τα δεδομένα από το αρχείο στον buffer file_buffer. Στον buffer αυτόν, το αρχείο είναι αποθηκευμένο ως μία ενιαία γραμμή. Σκοπός μας τώρα είναι να «σπάσουμε» αυτό το ενιαίο string σε επιμέρους ώστε να πάρουμε τις γραμμές του αρχείου αλλά και το πλήθος τους. Αυτό επιτυγχάνεται με την συνάρτηση strtok(), η οποία διαιρεί ένα string βάσει ενός χαρακτήρα delimiter που ορίζεται από εμάς. Εδώ χρησιμοποιούμε ως delimiter τον

χαρακτήρα αλλαγής γραμμής. Σε token που παίρνουμε από την strtok(), αφαιρούμε τα κενά πριν και μετά την γραμμή. Αυτό γίνεται χρησιμοποιώντας την συνάρτηση trimwhitespace().

```c
// Function to trim the whitespace from a string
// Idea from: https://stackoverflow.com/questions/122616/how-do-i-trim-leading-trailing-whitespace-in-a-standard-way
char *trimwhitespace(char *str){
    char *end;

    while (isspace((unsigned char)*str))
        str++;

    if (*str == 0)
        return str;

    end = str + strlen(str) - 1;

    while (end > str && isspace((unsigned char) *end))
        end--;

    return str;
}
```

Η συνάρτηση trimwhitespace - https://stackoverflow.com/questions/122616/how-do-i-trim-leading-trailing-whitespace-in-a-standard-way

Καθώς παίρνουμε νέες γραμμές, τις αντιγράφουμε στην δομή που θα χρησιμοποιεί το κυρίως πρόγραμμα για να εξυπηρετεί τους πελάτες, μετράμε τις γραμμές που εμφανίζονται και αυξάνουμε τον χώρο που καταλαμβάνει αυτή η δομή καθώς προχωράμε στο αρχείο με χρήση της συνάρτησης realloc().

```c
// Split the entire file
while (token != NULL){
    // Allocate space for the line. We use exactly the space needed
    // for each line
    file[lines] = malloc(strlen(token) * sizeof(char));
    // Copy the contents
    strcpy(file[lines], trimwhitespace(token));

    // Increment the line counter
    ++lines;

    // Realloc the file pointer, so it points to a region with one
    // char* space.
    file = realloc(file, (lines+1)*sizeof(char*));

    // Get the next token
    token = strtok(NULL, "\n");
}
```

Στο σημείο αυτό σημειώνεται πως οι μεταβλητές που ορίζονται στο κυρίως πρόγραμμα πριν την κλήση της fork(), αντιγράφονται στις θυγατρικές διεργασίες. Άρα, το πλήθος των γραμμών όπως και εκείνο των δοσοληψιών είναι ήδη γνωστό στις επιμέρους διεργασίες. Επομένως, δεν θα γίνει κάποια έξτρα ενημέρωση από την γονική σε αυτές.

4. Δημιουργία θυγατρικών διεργασιών

Η δημιουργία των θυγατρικών διεργασιών γίνεται σε μία απλή επανάληψη for.

```
// Start creating child processes. We create k_child processes
for (int i = 0; i < k_child; ++i){
    pid = fork();
    //The child process does nothing for now
    if (pid == 0)
        break;
}
```

Η fork() επιστρέφει 0 στην διεργασία-παιδί και το pid (process id) στην γονική διεργασία. Στο παραπάνω τμήμα κώδικα φαίνεται πως οι διεργασίες-παιδιά δεν κάνουν τίποτα παραπάνω στην επανάληψη, αντιθέτως βγαίνουν από αυτήν εκτελώντας την εντολή break.

5. Δέσμευση διαμοιραζόμενης μνήμης και συγχρονισμός μεταξύ διεργασιών

Για την επικοινωνία μεταξύ των διεργασιών παιδιών και της γονικής, πρέπει να οριστεί μια περιοχή μνήμης οι οποία θα μοιράζεται και από τις δύο. Κάθε διεργασία αποκτά τον δικό της χώρο διευθύνσεων μετά την fork, οπότε η κίνηση αυτή είναι απαραίτητη. Επίσης, για την απρόσκοπτη εκτέλεση το προγράμματος θα πρέπει οι διεργασίες να είναι συγχρονισμένες μεταξύ του, έτσι ώστε να αποφευχθούν περιπτώσεις όπου δύο διεργασίες θα γράφουν ταυτόχρονα στην διαμοιρασμένη μνήμη.

Για την διαμοιρασμένη μνήμη χρησιμοποιείται μια struct, η οποία αποτελείται από δύο buffers. Στον έναν γράφουμε οι θυγατρικές διεργασίες τον αριθμό της γραμμής που ζητούν από την γονική, ενώ στον άλλον γράφει η γονική την γραμμή του αρχείου. Η διαμοιραζόμενη μνήμη δεσμεύεται και προσαρτάται στο πρόγραμμα με τις κλήσεις συστήματος shmget() και shmat().

```
// Attach the memory to the procces. Check for possible errors
shmaddr = shmat(shmid, 0, 0);
if (shmaddr == (memory *) -1){
    perror("shmat: ");
    exit(EXIT_FAILURE);
}

// Initialize the 4 semaphores needed for synchronization. Check for errors
if ((semid = semget(IPC_PRIVATE, 4, IPC_CREAT | 0666)) == -1){
    perror("semget");
    exit(EXIT_FAILURE);
}
```

Για τον συγχρονισμό μεταξύ διεργασιών χρησιμοποιούνται σημαφόροι. Αυτοί ορίζονται στο κυρίως πρόγραμμα. Αρχικοποιούνται από την συνάρτηση sem_init() και ελέγχονται από τις συναρτήσεις sem_up και sem_down αντίστοιχα.

```c
// Initialize the 4 semaphores needed for synchronization. Check for errors
if ((semid = semget(IPC_PRIVATE, 4, IPC_CREAT | 0666)) == -1){
    perror("semget");
    exit(EXIT_FAILURE);
}

// Call sem_init function to initialize the semaphore values
sem_init(semid);
```

Δήλωση 4 σημαφόρων

```c
// Initializes the semaphores, described be sem_id
void sem_init(int sem_id){
    union semun arg0;
    union semun arg1;

    // Initial values for each semaphore
    arg0.val = 0;
    arg1.val = 1;

    // Sems #0 and #2 are initialized to 1 and #1 and #3 to 0
    // (arg1 and arg0 respectively)
    if (semctl(sem_id, 0, SETVAL, arg1) == -1){
        perror("semctl0: ");
        exit(EXIT_FAILURE);
    }

    if (semctl(sem_id, 1, SETVAL, arg0) == -1){
        perror("semctl1: ");
        exit(EXIT_FAILURE);
    }

    if (semctl(sem_id, 2, SETVAL, arg1) == -1){
        perror("semctl2: ");
        exit(EXIT_FAILURE);
    }

    if (semctl(sem_id, 3, SETVAL, arg0) == -1){
        perror("semctl3: ");
        exit(EXIT_FAILURE);
    }
}
```

Αρχικοποίησή τους με χρήση της συνάρτησης sem_init()

```c
// "Downs" the semaphore given
void sem_down(int sem_id, int sem_number){
    struct sembuf sem_oper;

    sem_oper.sem_num = sem_number;
    sem_oper.sem_op = -1;
    sem_oper.sem_flg = 0;

    if (semop(sem_id, &sem_oper, 1) != 0){
        perror("semop");
    }
}

// "Ups" the semaphore given
void sem_up(int sem_id, int sem_number) {
    struct sembuf sem_oper;

    sem_oper.sem_num = sem_number;
    sem_oper.sem_op = 1;
    sem_oper.sem_flg = 0;

    if (semop(sem_id, &sem_oper, 1) != 0){
        perror("semop");
    }
}
```

Συναρτήσεις ελέγχου των σημαφόρων

Χρησιμοποιούνται συνολικά 4 σημαφόροι. 2 για κάθε ενέργεια της κάθε οντότητας του προγράμματος (Read/Write και Client/Server).

6. Η λογική των client / server

Ο γονέας θα δεχτεί συνολικά αιτήματα ίσα με το γινόμενο των παιδιών επί τα αιτήματα που θα στείλει το καθένα. Έτσι, επαναλαμβάνεται η παρακάτω ενέργεια τόσες φορές.

Αρχικά, ο γονέας περιμένει κάποιο παιδί να γράψει στην κοινή περιοχή μνήμης που αντιστοιχεί στα αιτήματα προς αυτόν. Το «περιμένει» επιτυγχάνεται κάνοντας “down” την αντίστοιχη σημαφόρο. Η σημαφόρος αυτή έχει την τιμή 0 άρα η εκτέλεση του γονέα σταματάει μέχρι να γίνει “up” από το παιδί που αποστείλει κάποιο αίτημα. Όταν κάποιο αίτημα φτάσει, τότε ο γονέας συνεχίζει την εκτέλεσή του μετατρέποντας την γραμμή που έστειλε το παιδί από string σε int. Έπειτα, συνεχίζει κάνοντας “up” την σημαφόρο που επιτρέπει σε κάποιο άλλο παιδί να στείλει κάποιο αίτημα.

Στη συνέχεια, κάνει “down” την κατάλληλη σημαφόρο ώστε να αποκτήσει αποκλειστική πρόσβαση στην περιοχή μνήμης που γράφονται οι απαντήσεις από αυτόν και γράφει εκεί την ζητούμενη σειρά από το αρχείο.

Τέλος, αφότου ολοκληρωθεί η παραπάνω διαδικασία εξετάζει τις τιμές που επέστρεψαν οι διεργασίες παιδία και τερματίζει.

```
// The parent's code
if (pid > 0){
    int status;

    // The parent will receive children * transactions requests.
    // So the loop will go up to the product of these two
    for (int i = 0; i < k_child * n_trans; ++i){
        // Down the semaphore #1, so only the parent can access the shared memory
        sem_down(semid, 1);
        // Allocate a temporary string for the line number of the file requested to be saved
        char *temp = malloc(strlen(shmaddr->request_id) * sizeof(char));
        // Copy the contents from the shared memory to the temp string
        strcpy(temp, shmaddr->request_id);
        // Up the semaphore #0, so another child can write to the shared memory
        sem_up(semid, 0);

        // Down the semaphore #2, so only the parent can write to the shared memory
        sem_down(semid, 2);
        // Convert the line number from string to int
        int lid = atoi(temp);
        // Free the space taken up by temp
        free(temp);
        // Copy the line data from the "array" to the shared memory
        strcpy(shmaddr->response, file[lid]);
        // Up the semaphore #3
        sem_up(semid, 3);
    }

    // Check the return codes of all the processes
    for (int i = 0; i < k_child; ++i){
        wait(&status);
    }
}
```

Κώδικας διεργασίας γονέα

Οι θυγατρικές διεργασίες κάνουν της αντίστοιχη δουλεία, με την διαφορά ότι την επαναλαμβάνουν n_trans φορές. Οι διεργασίες αυτές παράγουν έναν τυχαίο αριθμό στο διάστημα και τον στέλνουν στην γονική. Περιμένουν (μέσω των σημαφόρων) την απάντηση από την γονική, μετρώντας τον χρόνο που έκανε η γονική να απαντήσει. Ο χρόνος μετράται με χρήση της συνάρτησης clock() και τις σταθεράς CLOCKS_PER_CYCLE. Ο χρόνος που εκτυπώνεται στην τυπική έξοδο είναι σε δευτερόλεπτα.

```c
// Child processes code
else {
    // Set the rand seed, according to each child's pid
    srand(getpid());

    double total_time = 0;
    // Repeat for n transactions
    for (int i = 0; i < n_trans; ++i){
        // Generate a random number in [0, lines)
        int request_id = rand() % (lines);
        // Convert the int to string. Idea taken from
        // https://stackoverflow.com/questions/8257714/how-to-convert-an-int-to-string-in-c
        int len = snprintf(NULL, 0, "%d", request_id);
        char *temp = malloc((len + 1) * sizeof(char));
        snprintf(temp, len + 1, "%d", request_id);

        // Down sem #0
        sem_down(semid, 0);
        // Hold the clock when the child starts the transaction
        clock_t begin = clock();
        // Copy the line number to the shared memory
        stpcpy((shmaddr->request_id), temp);
        // Free the allocated space for the line number
        free(temp);
        // Up sem #1
        sem_up(semid, 1);
        // Down sem #3
        sem_down(semid, 3);
        // Allocate space for the parent's response
        temp = malloc(strlen((shmaddr->response)) * sizeof(char));
        // Copy the contents from the shared memory
        strcpy(temp, shmaddr->response);
        // Print the response
        printf("Child[%d]Line %d -> %s\n", getpid(), request_id, temp);
        // Stop the count
        clock_t end = clock();
        // Add the delay to the total time sum. Used for the average calculation
        total_time += (double)(end - begin) / CLOCKS_PER_SEC;
        // Up sem #2
        sem_up(semid, 2);
    }
    // Print the child's stats
    printf("Child[%d]: Average wait time: %lf[s]\n", getpid(), total_time / n_trans);
    // Exit
    exit(EXIT_SUCCESS);
}
```

Κώδικας θυγατρικών διεργασιών