Προγραμματισμός Συστήματος

Εργασία 1

Completion report:

		ΝΑΙ	ΜΕΡΙΚΩΣ	ΟΧΙ
multijob.sh		X		
allJobsStop.sh		X		
	Εκκίνηση/τερματισμός/ εξαγωγή και έλεγχος ορθότητας εντολη από args		X	
Εκινηση Server	Ελεγχος αν υπαρχει και αν οχι τον ξεκινα	X		
Pipes	Ανοιγμα pipe	X		
	Ενημέρωση Server για αποστολή εντολής			
	αποστολή εντολής	X		
	λήψη απάντησης	X		
Γενικά	Ορθή εκκίνηση και δημιουργία jobExecutorServer.txt			
	Εκκίνησης ανοιγμα pipe /ληψη μηνύματος για προετοιμασια ληψης /λήψη εντολής / αποστολή απάντησης	X		
	υλοποίηση ουράς /εισαγωγή /εξαγωγή/ διαγραφή στοιχειου/ εκτύπωση όλων	X		
issueJob	Ανάθεση αναγνωριστικού σε εντολή και εισαγωγή στην ουρά	X		
	εκτελεση εντολων με fork	X		
	Διαχείριση SIGCHLD για εκκίνηση επόμενης εντολής από την ουρά	X		
setConcurrency	Έλεγχος εξαγωγή από ουρά κ εντολών ανάλογα με το consurency	X		
	Έλεγχος εκκίνησης νέων εντολών από την ουρά αν μεγάλωσε το consurency	X		

	Έλεγχος μη εκκίνησης νέων εντολών από την ουρά αν μίκρυνε το consurency			X
stop	Τερματισμός εντολής αν εκτελείται		X	
	Αφαίρεση εντολής από ουρά αν δεν εκτελείται	X		
poll running	Επιστροφή ορθών αποτελεσμάτων		X	
poll queued	Επιστροφή ορθών αποτελεσμάτων		X	
exit	Τερματισμος /αποστολή μηνύματος στο jobCommander / διαγραφή αρχείου		X	

Η Εργασία αποτελείται από τα εξής αρχεία:

1. Header Files (στο φάκελο include):

1.1. header.h: Περιέχει τους ορισμούς των βιβλιοθηκών που χρησιμοποιούνται από τα προγράμματα. Επίσης ορίζει τις gloabal μεατβλητές jobID_as_num και Concurrency καθώς και κάποιες συμβολικές σταθερές για τα ονόματα του αρχείου και των pipes.

1.2. Running_Queue.h: Περιέχει τους ορισμούς των εξης δομών:

1.2.1. Running_Queue η οποία αντιπροσωπεύει μια δομή ουράς για της διεργασίες που εκτελούνται

1.2.2. Running_Job_Info για τις πληροφορίες της καθεμίας από τις εν λόγω διεργασίες

1.2.3. Running_Queue_Node για την προσπέλαση των κόμβων της ουράς.

Επίσης περιέχει τους ορισμούς των συναρτήσεων που χρησιμοποιήθηκαν για την υλοποίηση της ουράς για τις δουλειές υπό εκτέλεση. Αυτές είναι οι παρακάτω:

- ***Running_Queue* r_queue_create(void);***

  Δημιουργεί και αρχικοποιεί μια καινούργια Running_Queue επιστρέφοντας δείκτη σε αυτή.

- ***bool is_r_queue_empty(Running_Queue* q);***

  Ελέγχει εάν η ουρά είναι άδεια και επιστρέφει true or false ανάλογα.

- ***void r_queue_insert(Running_Queue* q, Running_Job_Info* info);***

  Δημιουργεί και προσθέτει στο τέλος της ουράς q ένα καινούργιο κόμβο τύπου Running_Queue_node ο οποίος περιέχει τις πληροφορίες info για την καινούργια διεργασία υπό εκτέλεση.

- ***Running_Job_Info* r_queue_remove(Running_Queue* q);***

Αφαιρεί τον πρώτο κόμβο από την ουρά, ενημερώνοντας κατάλληλα τα πεδία position (θέση στην ουρά) των info των επόμενων κόμβων. Επιστρέφει δείκτη Running_Job_Info με τα στοιχεία της διεργασίας που αφαιρέθηκε.

- ***Running_Job_Info* r_queue_remove_ID(char* jobID, Running_Queue* q);***

  Αφαιρεί τον κόμβο που περιέχει τα στοιχεία της διεργασίας με ID το jobID από την ουρά, ενημερώνοντας κατάλληλα τα πεδία position (θέση στην ουρά) των info για τα jobs των επόμενων κόμβων από αυτόν που διαγράφτηκε. Επιστρέφει δείκτη Running_Job_Info με τα στοιχεία της διεργασίας που αφαιρέθηκε.

- **Running_Job_Info* r_queue_remove_pid(pid_t pid, Running_Queue* q);**

Αφαιρεί τον κόμβο που περιέχει τα στοιχεία της διεργασίας με αναγνωριστικό pid από την ουρά, ενημερώνοντας κατάλληλα τα πεδία position (θέση στην ουρά) των info για τα jobs των επόμενων κόμβων από αυτόν που διαγράφτηκε. Επιστρέφει δείκτη Running_Job_Info με τα στοιχεία της διεργασίας που αφαιρέθηκε.

- ***int r_queue_find_ID(char* jobID, Running_Queue* q);***

  Ψάχνει τον κόμβο που περιέχει τις πληροφορίες της διεργασίας με ID το jobID. Αν τη βρει επιστρέφει τη θέση της στην ουρά, αλλιώς -1.

- ***int r_queue_find_pid(pid_t pid, Running_Queue* q);***

  Ψάχνει τον κόμβο που περιέχει τις πληροφορίες της διεργασίας με αναγνωριστικό το pid. Αν τη βρει επιστρέφει τη θέση της στην ουρά, αλλιώς -1.

- ***Running_Queue_Node* r_queue_get_node(Running_Queue* q, int pos);***

  Επιστρέφει έναν δείκτη τύπου Running_Queue_Node που δείχνει στον κόμβο στη θέση pos της ουράς.

- ***void print_r_queue(Running_Queue* q);***

  Εκτυπώνει όλα τα στοιχεία της ουράς με τη σειρά. Στην εκτύπωση ο πιο αριστερός κόμβος είναι το τέλος της ουράς (rear) και ο πιο δεξής η αρχή της (front).

- ***void r_queue_destroy(Running_Queue* q);***

  Καταστρέφει την ουρά απελευθερώνοντας όση μνήμη είχε δεσμεύσει για τους κόμβους της, τις πληροφορίες που αποθηκεύονταν σε αυτούς κοκ.

1.3. Waiting_Queue.h: Περιέχει τους ορισμούς των εξής δομών:

1.3.1. Waiting _Queue η οποία αντιπροσωπεύει μια δομή ουράς για της διεργασίες που είναι στην ουρά αναμονής.

1.3.2. Waiting _Job_Info για τις πληροφορίες της καθεμίας από τις εν λόγω διεργασίες

1.3.3. Waiting _Queue_Node για την προσπέλαση των κόμβων της ουράς.

Επίσης περιέχει τους ορισμούς των συναρτήσεων που χρησιμοποιήθηκαν για την υλοποίηση της ουράς για τις δουλειές υπό εκτέλεση. Αυτές είναι οι παρακάτω:

- ***Waiting_Queue* w_queue_create(void);***

  Δημιουργεί και αρχικοποιεί μια καινούργια Waiting _*Queue επιστρέφοντας δείκτη σε αυτή.*

- ***bool is_w_queue_empty(Waiting_Queue* q);***

  Ελέγχει εάν η ουρά είναι άδεια και επιστρέφει true or false ανάλογα.

- ***void w_queue_insert(Waiting_Queue* q, Waiting_Job_Info* info);***

  Δημιουργεί και προσθέτει στο τέλος της ουράς q ένα καινούργιο κόμβο τύπου Waiting_Queue_node ο οποίος περιέχει τις πληροφορίες info για την καινούργια διεργασία η οποία τίθεται σε αναμονή.

- ***Waiting_Job_Info* w_queue_remove(Waiting_Queue* q);***

  Αφαιρεί τον πρώτο κόμβο από την ουρά, ενημερώνοντας κατάλληλα τα πεδία position (θέση στην ουρά) των info των επόμενων κόμβων. Επιστρέφει δείκτη Waiting_Job_Info με τα στοιχεία της διεργασίας που αφαιρέθηκε.

- ***Waiting_Job_Info* w_queue_remove_ID(char* jobID, Waiting _Queue* q);***

  Αφαιρεί τον κόμβο που περιέχει τα στοιχεία της διεργασίας με ID το jobID από την ουρά, ενημερώνοντας κατάλληλα τα πεδία position (θέση στην ουρά) των info για τα jobs των επόμενων κόμβων από αυτόν που διαγράφτηκε. Επιστρέφει δείκτη Running_Job_Info με τα στοιχεία της διεργασίας που αφαιρέθηκε.

- ***int w_queue_find_ID(char* jobID, Waiting_Queue* q);***

  Ψάχνει τον κόμβο που περιέχει τις πληροφορίες της διεργασίας με ID το jobID. Αν τη βρει επιστρέφει τη θέση της στην ουρά, αλλιώς -1.

- ***Waiting_Queue_Node* w_queue_get_node(Waiting_Queue* q, int pos);***

  Επιστρέφει έναν δείκτη τύπου Waiting _Queue_Node που δείχνει στον κόμβο στη θέση pos της ουράς.

- ***void print_w_queue(Waiting_Queue* q);***

  Εκτυπώνει όλα τα στοιχεία της ουράς με τη σειρά. Στην εκτύπωση ο πιο αριστερός κόμβος είναι το τέλος της ουράς (rear) και ο πιο δεξής η αρχή της (front).

- ***void w_queue_destroy(Waiting_Queue* q);***

  Καταστρέφει την ουρά απελευθερώνοντας όση μνήμη είχε δεσμεύσει για τους κόμβους της, τις πληροφορίες που αποθηκεύονταν σε αυτούς κοκ.

1.4. JES_helping_functions.h: Περιέχει τους ορισμούς όλων των συναρτήσεων που διευκολύνουν την λειτουργία του jobExecutorServer. Αυτές είναι οι εξής:

1.4.1. ***char* convert_jobID_to_string();***

Δημιουργεί ένα μοναδικό αναγνωριστικό διεργασίας μεβάση τη global μεταβλητή jobID_as_num και επιστρέφει δείκτη σε αυτό. Επίσης αυξάνει το jobID_as_num κατά 1.

1.4.2. ***char* format_w_job_info(Waiting_Job_Info* job_info);***

Δημιουργεί το μήνυμα εκτύπωσης που θα επιστραφεί στο χρήστη μετά την επιτυχή εισαγωγή μιας διεργασίας στην ουρά εκτέλεσης.

1.4.3. ***char* format_r_job_info(Running_Job_Info* job_info);***

Δημιουργεί το μήνυμα εκτύπωσης που θα επιστραφεί στο χρήστη μετά την επιτυχή εισαγωγή μιας διεργασίας στην ουρά αναμονής.

1.4.4. ***Waiting_Job_Info* issue_waiting_job(int argc,char* argv[], Waiting_Queue* waiting_queue);***

Προσθέτει μια διεργασία στην ουρά αναμονής, φροντίζοντας να αποθηκεύει τις κατάλληλες πληροφορίες στο πεδίο job_info της διεργασίας έτσι ώστε αυτή να μπορεί μετα να εκτελεστεί.

1.4.5. ***Running_Job_Info* issue_running_job(Waiting_Job_Info* waiting_info,char* argv[], Running_Queue* running_queue);***

Προσθέτει μια διεργασία στην ουρά εκτέλεσης και καλεί μια συνάρτηση από την οικογένεια των exec για να την εκτελέσει, αποθηκεύοντας το pid της. Όταν η εργασία εκτελείται έχοντας πρότινος υπάρξει στην ουρά αναμονής, ο δείκτης argv είναι NULL και χρησιμοποιείται η δομή waiting_info για την αποθήκευση των πληροφοριών της διεργασίας. Αντιθέτως εάν η διεργασία δεν περίμενε να εκτελεστεί και συνεπώς δεν υπήρχε στην ουρά αναμονής ο δείκτης waiting_info είναι NULL, και οι πληροφορίες λαμβάνονται από το argv.

1.4.6. ***int set_concurrency(int new_concurrency, Waiting_Queue* waiting_queue, Running_Queue* running_queue);***

Θέτει καινούργιο βαθμό παραλληλίας για την εκτέλεση διεργασιών. Εάν ο καινούργιος βαθμός είναι μικρότερος από τον παλιο δεν διακοπτει τις διεργασίες που τρέχουν μέχρι να τελειώσουν. Εάν ο καινούργιος βαθμός είναι μεγαλύτερος από τον παλιό υπολογίζει πόσες διεργασίες μπορούν να ξεκινήσουν να εκτελούνται (με βαση το νέο concurrency και το μέγεθος της ουρας εκτέλεσης) και ξεκινά να της εκτελεί με χρήση issue_running_job.

1.4.7. ***char* poll_running(Running_Queue* running_queue);***

Εκτυπώνει πληροφορίες για όλες τις διεργασίες στην ουρά εκτέλεσης.

1.4.8. ***char* poll_waiting(Waiting_Queue* waiting_queue);***

Εκτυπώνει πληροφορίες για όλες τις διεργασίες στην ουρά αναμονής.

1.4.9. ***int stop_job(char* jobID,Running_Queue* running_queue, Waiting_Queue* waiting_queue);***

Ένα η διεργασία με αναγνωριστικο jobID τρέχει, δηλαδή βρίσκεται στην ουρά εκτέλεσης, χρησιμοποιεί SIGKILL για να την τερματίσει και επιστρέφει 0. Εάν η διεργασία είναι στην ουρά αναμονής, την αφαιρεί και επιστρέφει 1. Εάν δεν βρέθηκεσε καμία ουρά επιστρέφει -1.

1.4.10. ***void jc_to_jes_sig_handler(int sig, siginfo_t* sig_info, void* context);***

Καλείται από τον jobExecutorServer και διαχειρίζεται το input από τον jobCommander. Είναι υπεύθυνος για το άνοιγμα των pipes από την πλευρά του jobExecutorServer και για την κληση των κατάλληλων συναρτήσεων (βλ. πάνω) ανάλογα με την εντολή/εντολές που έδωσε ο χρήστης. Είναι επίσης υπεύθνος για το σχηματισμό του μηνύματος που θα επιστραφεί στον jobCommander μετα την εκτέλεση των εντολών (για σκοπούς ομοιομορφίας επιστρέφεται πάντα ένα μηνυμα στον χρήστη, ακόμα και στην περίπτωση εντολής setConcurrency) και για το κλείσιμο του jobExecutorServer.txt αρχείου μετά τη λήψη της εντολής exit.

1.4.11. ***void sigchild_handler(int sig);***

Διαχειρίζεται την περίπτωση όπου μια διεργασία τελειώνει (μόνη της). Αφού βρεί ποια διεργασία από την ουρά εκτέλεσης έστειλε το μήνυμα τερματισμού, την αφαιρεί από την εκεί και στη συνέχεια φροντιζει να θέσει υπο εκτέλεση όσες διεργασίες ακόμα επιτρέπει το concurrency (εάν υπάρχουν σε αναμονη).

2. Source *Files (στο φάκελο src):*

2.1.1. jobCommander.c

Λαμβάνει το input από το χρήστη και κάνει έλεγχο για ελλιπή ή λάθος εντολές. Αν οι εντολές είναι σωστές ελέγχει εάν τρέχει ήδη ο jobExecutorServer. Εάν όχι το καλεί με χρήση fork, αλλιώς ανοίγει ένα pipe για διάβασμα, διαβάζει το pid του jobExecutorServer και το ξανακλείνει. Στη συνέχεια ανοίγει ένα pipe για γράψιμο, ξεκινάει να γράφει το input σε αυτό, και όταν τελειώνει το ξανακλείνει. Κάνει sleep για ένα δευτερόλεπτο περιμένοντας να ετοιμαστεί η απάντηση από τον jobExecutorServer και στη συνέχεια ανοίγει πάλι ένα pipe για διάβασμα, την διαβάζει και εκτυπώνει το μήνυμα στην κονσόλα. Τέλος κλείνει τα pipes και τερματίζει.

2.1.2. jobExecutorServer.c

Αρχικοποιεί δύο άδειες δομές ουράς για τις διεργασίες υπο εκτέλεση και αυτές σε αναμονή. Έπειτα δημιουργεί το αρχείο jobExecutorServer.txt, το ανοίγει, και γράφει το pid του εαυτού του σε αυτό έτσι ώστε να μπορεί να το διαβάσει ο jobCommander. Στη συνέχεια αρχικοποιεί τις δομές και τις συναρτήσεις που θα χρησιμοποιηθούν για τη διαχείριση του σήματος από τον jobCommander στον jobExecutorServer, καθώς και για το σήμα που στέλνει η κάθε διεργασία παιδί στο γονιο της (jobExecutorServer) όταν τελειώνει. Τέλος ξεκινά έναν ατέρμων βρόγχο για να παραμείνει ο server ανοιχτος στο backtound καθώς η λειτουργία του προγράμματος θα λήξει από τον

jc_to_jes_sig_handler που αναφέρθηκε και νωρίτερα, όταν αυτός λάβει την εντολή τερματισμού.

2.1.3. Running_Queue.c

Υλοποίηση της ουράς εκτέλεσης. Υπάρχει μια main συνάρτηση η οποία είναι commented out η οποία χρησιμοποιήθηκε για testing των λειτουργιών της ουράς.

2.1.4. Waiting_Queue.c

Υλοποίηση της ουράς αναμονής. Υπάρχει μια main συνάρτηση η οποία είναι commented out η οποία χρησιμοποιήθηκε για testing των λειτουργιών της ουράς.

2.1.5. Makefile

Κάποιες χρήσιμες εντολές είναι οι εξης:

- *make all*: Κάνει compile τα jobExecutorServer και jobCommander. Υπάρχουν σε σχόλια και τα Running_Queue, Waiting_Queue γιατι εκτελούταν ατόμικα στη φάση του testing. Χρειάζεται να βγεί από τα σχόλια η υλοποίηση της main στο καθένα για να λειτουργήσει.
- make clean: Διαγράφει όλα τα αρχεία ου δημιουργήθηκαν από πορηγούμενες εκτελέσεις.
- make new: Κάνει make clean και *make all*. Μετα την εκτέλεση του το πρόγραμμα είναι έτοιμο να τρέξει από τη αρχή.
- make valgrind: Τρέχει την εντολη valgrind --leak-check=full --show-leak-kinds=all ./jobExecutorServer. Το παραπάνω χρησιμοποιήθηκε για εκτέλεση του jobExecutorServer σε διαφορετικό παράθυρο έτσι ώστε να μπορεί να παρατηρηθεί καλύτερα η συμπεριφορά του καθώς και τυχόν memory leaks.
- make jobs: κάνει compile τα βοηθητικά προγράμματα που βρίσκονται στο φάκελος test_programs

2.1.6. Φάκελος test_programs

- Εκεί βρίσκονται βοηθητικά αρχεία για δοκιμή στην εφαρμογη, όπως το hello_world, even_odd και progDelay.
- Επίσης εκεί βρίσκονται τα bash scrips που ζητούνται, τα multijob.sh και allJobsStop.sh

Παραδοχές/Ιδιαιτερότητες:

1. Στην υλοποίηση της ουράς για τις Running_Queue και Waiting_Queue, θεώρησα ότι η ουρά μεγαλώνει προς τα αριστερά (όπως κοιτάμε την οθόνη). Δηλαδή ο πιο δεξής της κόμβος είναι το queue front και ο πιο αριστερός το queue rear. Κατά συνέπεια στη δομή που αντιπροσωπεύει έναν κόμβο της ουράς, και στις δύο περιπτώσεις, το πεδίο next αναφέρεται στον κόμβο μια θέση στα δεξιά του δικού μας και το previous σε αυτόν μια θέση αριστερά.

2. Oi theseis sthn oyra ksekinane apo 1 anti gia 0
3. Όπως σχολιάστηκε και πριν ο jobExecutorServer επιστρέφει πάντα ένα μήνυμα στον jobCommander ακόμα και μετά την εντολή setConcurrency που δεν ζητείται, για λόγους ομοιομορφίας και διευκόλυνσης.

Παρόλου που οι ουρές που χρησιμοποιούνται είναι global, στέλνονται κανονικά στις συναρτήσεις που τις χρησιμοποιούν ως ορίσματα καθώς αυτό κάναι τον κώδικα πιο modular.

Υπάρχουν κάποια printf καθώς και κάποιες συναρτήσεις main (στα Running_Queue, Waiting_Queue αρχεία) οι οποίες χρησιμοποιήθηκαν για testing τις οποίες άφησα σε σχόλια σε περίπτωση που διευκολύνουν την διόρθωση

```
Tsekare      free(node->job_info->parameters);
```