ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Κ24: Προγραμματισμός Συστήματος – Εαρινό Εξάμηνο 2014
2η Προγραμματιστική Εργασία
Ημερομηνία Υποβολής: 15/06/2014

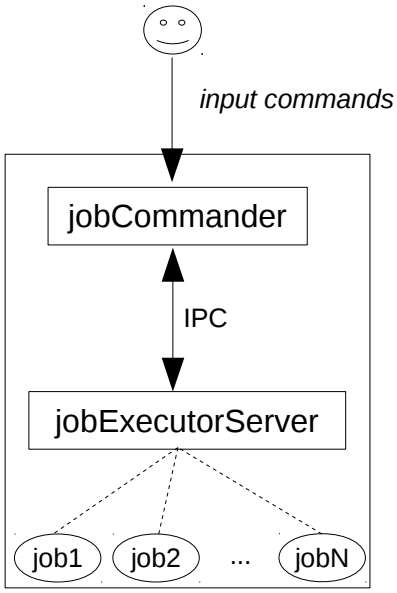
Στόχος αυτής της εργασίας είναι η εξοικείωσή σας με τις κλήσεις συστήματος της οικογένειας exec*(), της fork(), όπως επίσης και με κλήσεις συστήματος χαμηλού επιπέδου για τη διαχείριση αρχείων και το bash scripting.
Θα υλοποιήσετε ένα πρόγραμμα που έχει ως σκοπό την εκτέλεση εργασιών που λαμβάνει από κάποιο αρχείο εισόδου. Το πρόγραμμα αυτό θα προσφέρει στο χρήστη πληροφορία αλλά και τη δυνατότητα να διαχειριστεί το σύνολο των υπό εκτέλεση διεργασιών. Στη συνέχεια, θα γράψετε μία σειρά από bash scripts τα οποία θα προσφέρουν στο χρήστη πιο σύνθετες δυνατότητες.

Εφαρμογή jobExecutor
Αναπτύξτε την εφαρμογή jobExecutor η οποία θα αναλαμβάνει την εκτέλεση εργασιών (jobs) που θα δίνονται από το χρήστη. Η δουλειά του jobExecutor είναι να βάζει εργασίες (jobs) να τρέξουν, αλλά με έλεγχο του ρυθμού ροής τους (flow control). Για το λόγο αυτό, βασική έννοια είναι ο τρέχων βαθμός παραλληλίας (concurrency). Για παράδειγμα, ας θεωρήσουμε ότι ο βαθμός παραλληλίας είναι 4. Αυτό σημαίνει ότι οι 4 πρώτες διαθέσιμες εργασίες που έχουν υποβληθεί στην εφαρμογή (ή όσες υπάρχουν, αν έχουν υποβληθεί λιγότερες από 4) θα πρέπει να εκτελούνται, με τις υπόλοιπες σε ουρά αναμονής. Οι εργασίες σε ουρά αναμονής ΔΕΝ έχουν process στο λειτουργικό σύστημα: ο jobExecutor πολύ απλά δεν έχει ξεκινήσει να τις εκτελεί ακόμα. Αν μία εργασία από τις 4 τερματίσει, τότε ο jobExecutor θα αρχίσει να εκτελεί την πρώτη από την ουρά αναμονής. Αν ο βαθμός παραλληλίας αλλάξει, ο jobExecutor θα προσαρμοστεί αντίστοιχα, χωρίς να σταματήσει εργασίες που ήδη τρέχουν, σε περίπτωση μείωσης του βαθμού παραλληλίας.
Η εφαρμογή (jobExecutor) θα προσφέρει τη λειτουργικότητά της στο χρήστη μέσω δύο διεργασιών. Η πρώτη διεργασία (jobCommander) δίνει τη δυνατότητα στο χρήστη να αλληλεπιδράσει με την εφαρμογή μέσω απλών εντολών τις οποίες επικοινωνεί στη δεύτερη διεργασία (jobExecutorServer), η οποία ουσιαστικά υλοποιεί τον πυρήνα της λειτουργικότητας της εφαρμογής. Στο Σχήμα 1 παρουσιάζεται μία αναπαράσταση της δομής της εφαρμογής.
Ο χρήστης μπορεί να αλληλεπιδράσει με την εφαρμογή καλώντας το πρόγραμμα jobCommander και περνώντας του σαν παράμετρο μία από τις εξής εντολές:

1. issuejob <job>: Μέσω αυτής της εντολής εισάγονται εργασίες στο σύστημα που θέλουν να εκτελεστούν. Το job είναι μια συνηθισμένη γραμμή εντολών Unix. Η εφαρμογή αναθέτει ένα μοναδικό αναγνωριστικό (ζεύγος <jobID,job>) στην

εργασία, το οποίο και το επιστρέφει, μαζί με το αν είναι υπό εκτέλεση ή όχι.

2. setConcurrency <N>: Η παράμετρος αυτή θέτει το βαθμό παραλληλίας, δηλαδή το μέγιστο αριθμό ενεργών εργασιών που μπορεί να εκτελεί ανά πάσα χρονική στιγμή η εφαρμογή (δεδομένου ότι υπάρχουν διαθέσιμες). Η προκαθορισμένη τιμή είναι 1.
3. stop <jobID>: Τερματίζεται η εκτέλεση της εργασίας με το συγκεκριμένο αναγνωριστικό. Σε περίπτωση που δεν εκτελείται, αλλά είναι υποψήφια για μελλοντική εκτέλεση, αφαιρείται από την ουρά των υπό εκτέλεση εργασιών.
4. poll [running,queued]: Για κάθε εργασία που είναι υπό εκτέλεση (running) αυτή τη χρονική στιγμή ή, για κάθε εργασία που είναι σε κατάσταση αναμονής (queued), επιστρέφει το ζεύγος <jobID,job> το οποίο και παρουσιάζεται στο χρήστη. Η επιστρεφόμενη πληροφορία εξαρτάται από την τιμή του 2ου ορίσματος.
5. exit: Τερματίζεται η λειτουργία του jobExecutorServer.



Σχήμα 1: Σχηματική αναπαράσταση της αρχιτεκτονικής του jobExecutor.

Ο κύκλος ζωής του jobCommander είναι μία μόνο εντολή. Δηλαδή, η εκτέλεσή του ολοκληρώνεται όταν μεταβιβάσει με επιτυχία την εντολή στον jobExecutorServer.

Μέθοδος Υλοποίησης

Κάθε φορά που εκτελείται ένας jobCommander πρέπει να είναι σε θέση να αναγνωρίσει αν ο jobExecutorServer είναι ενεργός ή όχι. Για παράδειγμα, αυτό θα μπορούσε να επιτευχθεί εξετάζοντας την ύπαρξη ενός αρχείου (με προσυμφωνημένο όνομα) που θα δημιουργεί ο δεύτερος και θα περιέχει το αναγνωριστικό της διεργασίας (process id/pid) του server. Σε περίπτωση που δεν είναι ενεργός ο jobExecutorServer, θα πρέπει να τον δημιουργήσει ο jobCommander.

Η επικοινωνία ανάμεσα στις δύο διεργασίες λαμβάνει χώρα μέσω named pipes (κλήση

συστήματος mkfifo()). Η προκαθορισμένη συμπεριφορά των named pipes είναι να μπαίνει σε κατάσταση αναμονής η διεργασία που ανοίγει το ένα άκρο μέχρι να ανοιχτεί η σωλήνωση και από το άλλο άκρο. Βέβαια, μπορούμε να αποφύγουμε την παραπάνω συμπεριφορά αν θέσουμε το O_NONBLOCK flag στο δεύτερο όρισμα της κλήσεως συστήματος open(). Για παράδειγμα, αν θέλουμε να ανοίξουμε ένα named pipe για ανάγνωση χωρίς να τεθούμε σε αναμονή, κάνουμε την κλήση open(pipe_name , O_RDONLY | O_NONBLOCK). Προσοχή, αν σε αυτήν την περίπτωση ανοιχτεί το named pipe για γράψιμο (από το ένα άκρο) και δεν είναι ανοιχτό για διάβασμα (από το άλλο άκρο), τότε θα επιστραφεί σφάλμα. Είστε ελεύθεροι να διαλέξετε όποια μέθοδο λειτουργίας των σωληνώσεων θέλετε.
Ο jobCommander πρέπει με κάποιον τρόπο να ενημερώνει τον jobExecutorServer ότι σκοπεύει να του μεταβιβάσει κάποια εντολή μέσω της σωλήνωσης έτσι ώστε ο δεύτερος να πάει να τη διαβάσει. Αυτό, για παράδειγμα, θα μπορούσε να επιτευχθεί με την αποστολή κάποιου προσυμφωνημένου σήματος (signal-software interrupt).
Ο jobExecutorServer αρχικά είναι αδρανής μέχρι να δεχτεί κάποια/ες εντολή/ες issuejob, όπου ξεκινά την εκτέλεσή τους, λαμβάνοντας υπόψη τον βαθμό παραλληλίας. Για κάθε εργασία που θέλει να εκτελέσει, της αναθέτει ένα μοναδικό αναγνωριστικό, δημιουργεί μία διεργασία-παιδί μέσω της κλήσεως συστήματος fork() η οποία αμέσως εκτελεί την εργασία μέσω κάποιας εκ των συναρτήσεων της οικογένειας exec*() (execv(),execvp κλπ.). Στην περίπτωση που η εργασία δεν γίνεται να εκτελεστεί σε αυτή τη χρονική στιγμή, εισάγεται σε μία ουρά αναμονής με σκοπό να εκτελεστεί όταν θα έρθει η σειρά της. Σε κάθε περίπτωση, επιστρέφεται στο χρήστη η τριπλέτα <jobID,job,STATUS> που περιέχει το μοναδικό αναγνωριστικό που έχει ανατεθεί στην εργασία, καθώς και την κατάσταση στην οποία βρίσκεται (υπό εκτέλεση ή αναμονή).
Κατά την ολοκλήρωση μίας διεργασίας-παιδί, ο πυρήνας του Linux αποστέλλει στην γονική της διεργασία το σήμα (signal) SIGCHLD. Ο server σας θα πρέπει να διαχειρίζεται αυτό το σήμα έτσι ώστε να ενημερωθεί, σε περίπτωση που όταν το λάβει χειρίζεται κάποιο άλλο ασύγχρονο γεγονός (π.χ. το προσυμφωνημένο σήμα που αναφέρθηκε προηγουμένως για την επικοινωνία μέσω της σωλήνωσης με τον jobCommander), για τον τερματισμό κάποιας διεργασίας-παιδί. Όταν γίνει αυτό, θα πρέπει να πάρει από την ουρά αναμονής την επόμενη εργασία και να τη θέσει υπό εκτέλεση (δεδομένου ότι υπάρχει εργασία διαθέσιμη).
Συνολικά, ο jobExecutorServer πρέπει να επικοινωνεί με τον jobCommander μέσω named pipe καθώς και να ενημερώνεται για τον τερματισμό των διεργασιών-παιδιών του μέσω του σήματος (signal) SIGCHLD. Οι μηχανισμοί υλοποίησης όλης της υπόλοιπης λειτουργικότητας είναι δική σας επιλογή.

Bash Scripting

Αναπτύξτε τα ακόλουθα bash scripts, τα οποία έχουν ως σκοπό να χρησιμοποιήσουν την απλή διεπαφή του jobExecutor με σκοπό να παρέχουν στο χρήστη λειτουργίες υψηλότερου επιπέδου:

1. multijob.sh <file1> <file2> ... <fileN>: Το script αυτό λαμβάνει σαν είσοδο

αρχεία τα οποία περιέχουν εργασίες προς εκτέλεση, μία ανά γραμμή. Σκοπός είναι να εισαχθούν όλες αυτές οι εργασίες στο σύστημα.

2. allJobsStop.sh: Το script αυτό σταματά όλες τις εργασίες, είτε είναι υπό εκτέλεση είτε όχι, που υπάρχουν ενεργές στον jobExecutor.

Διαδικαστικά:

- Το πρόγραμμά σας θα πρέπει να τρέχει στα μηχανήματα Linux/Unix της σχολής.
- Ονόματα και e-mail βοηθών θα βρείτε στην ιστοσελίδα του μαθήματος.
- Για επιπρόσθετες ανακοινώσεις, παρακολουθείτε το forum του μαθήματος στο piazza.com για να δείτε ερωτήσεις/απαντήσεις/διευκρινίσεις που δίνονται σχετικά με την άσκηση. Η παρακολούθηση του forum στο piazza.com είναι ***υποχρεωτική***.

Τι πρέπει να παραδοθεί:

- Όλη η δουλειά σας σε ένα tar-file που να περιέχει όλα τα source files, header files, Makefile. **ΠΡΟΣΟΧΗ**: φροντίστε να τροποποιήσετε τα δικαιώματα αυτού του αρχείου πριν την υποβολή, π.χ. με την εντολή chmod 755 OnomaEponymoProject1.tar (περισσότερα στη σελίδα του μαθήματος).
- Μια σύντομη περιγραφή (1–2 σελίδες) για τις επιλογές που κάνατε στο σχεδιασμό της άσκησης, σε μορφή PDF.
- Οποιαδήποτε πηγή πληροφορίας, συμπεριλαμβανομένου και κώδικα που μπορεί να βρήκατε στο Διαδίκτυο θα πρέπει να αναφερθεί και στον πηγαίο κώδικά σας αλλά και στην παραπάνω αναφορά.

Τι θα βαθμολογηθεί:

- Η συμμόρφωση του κώδικά σας με τις προδιαγραφές της άσκησης.
- Η οργάνωση και η αναγνωσιμότητα (μαζί με την ύπαρξη σχολίων) του κώδικα.
- Η χρήση Makefile και η κομματιαστή σύμβολο-μετάφραση (separate compilation).
- Η αναφορά που θα γράψετε και θα υποβάλετε μαζί με τον πηγαίο κώδικα σε μορφή PDF.

Άλλες σημαντικές παρατηρήσεις:

- Οι εργασίες είναι **ατομικές**.
- Όποιος υποβάλλει / δείχνει κώδικα που δεν έχει γραφτεί από τον/την ίδιο/ίδια μηδενίζεται στο μάθημα.
- Αν και αναμένεται να συζητήσετε με φίλους και συνεργάτες το πως θα επιχειρήσετε να δώσετε λύση στο πρόβλημα, ***αντιγραφή κώδικα*** (οποιασδήποτε μορφής) είναι κάτι που ***δεν επιτρέπεται*** και δεν πρέπει να γίνει. Οποιοσδήποτε βρεθεί αναμεμειγμένος σε αντιγραφή κώδικα ***απλά παίρνει μηδέν*** στο μάθημα. Αυτό ισχύει για όλους όσους εμπλέκονται, ανεξάρτητα από το ποιος έδωσε/πήρε κλπ.
- Το πρόγραμμά σας θα πρέπει να γραφτεί σε C (ή C++ χωρίς όμως STL extensions).