Λειτουργικά Συστήματα Υπολογιστών

$1^{η}$ Εργαστηριακή Άσκηση

Κλήσεις Συστήματος, Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

Μπίλιας Ευστάθιος Γκούντης Βασίλης

23 Απριλίου 2024

ΣΗΜΜΥ
Εθνικό Μετσόβιο Πολυτεχνείο

Περιεχόμενα

1 Ανάγνωση και εγγραφή αρχείων στη C και με τη βοήθεια κλήσεων συστήματος

Για την υλοποίηση της πρώτης άσκησης ξεκινήσαμε να διαβάζουμε τα manuals των εντολών που μας δοθήκαν ως βοηθήματα. Για την περαιτέρω κατανόηση τους, μελετήσαμε τις διαφάνειες και τα παραδείγματα του μαθήματος και αφού φτάσαμε σε σημείο να καταλαβαίνουμε πως λειτουργούν οι εντολές **open, read, write** και **close** ξεκινήσαμε να γράφουμε τον κώδικα μας ακολουθώντας τα παραδείγματα.

1.1 Open

Αρχικά, χρησιμοποιήσαμε την **open** για να ανοίξουμε τα αρχεία τα οποία μας δίνει ο χρήστης από τη γραμμή εντολών και τις μεταβλητές **fpr** και **fpw** στις οποίες αποθηκεύεται η τιμή του **file descriptor** που επιστρέφει η **open** ώστε να διαχωρίσουμε το αρχείο ανάγνωσης από το αρχείο εγγραφής.

1.2 Read

Στη συνέχεια, κάνουμε την ανάγνωση του αρχείου μέσω της χρήσης της **read**, η οποία παίρνει σαν πρώτο όρισμα το **fpr**, που περιέχει τον **file descriptor** του αρχείου που θέλουμε να διαβάσουμε, σαν δεύτερο όρισμα έναν **buffer** που έχουμε δηλώσει προηγουμένως και στον οποίο θα πραγματοποιηθεί η ανάγνωση και σαν τρίτο όρισμα το μέγεθος που θέλουμε να διαβάσουμε, στην περίπτωση μας θέλουμε να διαβάσουμε όσο χωράει ο **buffer**. Την **read** χρησιμοποιούμε μέσα σε ένα **for loop** το οποίο τρέχει ασταμάτητα μέχρι να συναντήσει την **break**, την οποία συναντάει όταν ο τύπος **rcnt** γίνει 0 (μηδέν), δηλαδή όταν έχει τελειώσει η ανάγνωση του αρχείου. Μέσα στη **read** υλοποιούμε επίσης και το κομμάτι του κώδικα το οποίο ευθύνεται για την καταμέτρηση του χαρακτήρα που έχουμε ζητήσει στη γραμμή εντολών. Για να επιτευχθεί αυτό, μεσα σε ένα **for loop** το οποίο εκτελείται από 0 μέχρι **rcnt** δηλαδή μέχρι να διατρέξει όλα τα στοιχειά του **buffer** ελέγχεται αν η τιμή στην θέση **i** του πίνακα **buff** είναι ίση με τον χαρακτήρα **sc** τον οποίο ψάχνουμε. Κάθε φορά που θα εντοπίζεται ο χαρακτήρας θα ανεβαίνει και η μεταβλητή **count** την οποία έχουμε αρχικοποιήσει στο 0.

1.3 Write

Παρακάτω, δημιουργούμε έναν άλλο πίνακα με το όνομα **message** στον οποίο τοποθετούμε το μήνυμα που θέλουμε να εμφανίζεται στο αρχείο εγγραφής. Και μέσα σε ένα **while loop** χρησιμοποιούμε την **write** η οποία παίρνει σαν όρισμα το **file descriptor** του αρχείου εγγραφής καθώς και το μήνυμα που εμφανίζεται στον πίνακα **message** ώστε να το γράψει μέσα στο αρχείο. Επιπλέον, χρησιμοποιούμε τις **len** και **idx** για να μας βοηθήσουν στην εγγραφή, η **len** περιέχει το μήκος του μηνύματος και η **idx** χρησιμοποιείται σαν κέρσορας ώστε να γνωρίζουμε σε ποιο μέρος του μηνύματος βρισκόμαστε.

1.4 Close

Τέλος, για να ολοκληρώσουμε χρησιμοποιούμε την **close** ώστε να κλείσουμε τα αρχεία που είχαμε ανοίξει προηγουμένως.

2 Δημιουργία Διεργασιών

2.1 Greetings

Στο πρώτο ερώτημα της δεύτερης άσκησης το πρόγραμμα ξεκινά με μια κύρια διεργασία (πατέρας) που δημιουργεί μια νέα διεργασία (παιδί) με την κλήση **fork**.

2.1.1 Διεργασία Παιδί

Αν η **fork** επιστρέψει 0, βρισκόμαστε στη διεργασία παιδιού, όπου καλούνται οι **getpid** και **getppid** για να πάρουν τα **PID** της διεργασίας παιδιού και του πατέρα αντίστοιχα. Στη συνέχεια, το παιδί καλεί τη συνάρτηση **child** για να τυπώσει ένα μήνυμα με αυτές τις πληροφορίες.

2.1.2 Διεργασία Πατέρας

Αν η **fork** επιστρέψει θετικό αριθμό, βρισκόμαστε στη διεργασία πατέρα. Ο πατέρας καλεί τη συνάρτηση **father** με το **PID** της νέας διεργασίας (παιδιού) και περιμένει το παιδί να τερματίσει με την κλήση **wait**. Μετά, ο πατέρας τυπώνει ένα μήνυμα που δείχνει το αποτέλεσμα της αναμονής χρησιμοποιώντας τη συνάρτηση **explain_wait_status**.

2.1.3 Error

Αξίζει να σημειώθει οτάν η fork επιστρέφει -1 αποτέλει σφαλμά.

2.1.4 Βοηθητική συνάρτηση explain_wait_status

Την συνάρτηση **explain_wait_status** την πήραμε ετοιμή από τις διαφάνειες και την χρησιμοποιήσαμε για να παρατηρήσουμε πως συμπεριφέρονται τα παιδιά διεργασίες.

2.2 Μεταβλητή x

Αρχικά θέτουμε μια μεταβλητή $x = 0$ στον πατέρα (**πρίν κάνουμε την fork**). Ύστερα μέσα στο παιδί αναθέτουμε την τιμή του x σε 1, ενώ μέσα στον πατέρα (αυτή τη φορά **μετά την fork** και όταν αυτή επιστρέφει θετικό αριθμό) την αναθέτουμε σε 2. Αυτό που παρατηρούμε είναι ότι το παιδί εκτυπώνει ότι $x = 1$ και ο πατεράς ότι $x = 2$. Με λίγα λόγια συμπεραίνουμε ότι κανένας από τους δύο δεν εκτυπώνει $x = 0$.

2.3 Αναζήτηση χαρακτήρα από το παιδί

Η φιλοσοφία του κωδικά είναι παρομοία με του κώδικα του πρώτου ερωτημάτος. Η μόνη ουσιαστική διαφορά είναι οτί τα read και write, όπως και η καταμέτρηση του χαρακτήρα θα γίνουν μόνο από το παιδί (δηλαδή μετά την fork και οτάν αυτή επιστρέφει 0), ενώ ο πατέρας αναλαμβάνει αποκλειστικά τα open και close (μετά την fork, αλλά αυτή την φορά οτάν επιστρέφει θετικό αριθμό). Επίσης, χρησιμοποιούμε την wait ώστε ο πατέρας να περιμένει το παιδί να τερματίσει πρίν κλείσει τα αρχεία.

2.4 Χρήση execv για εκτέλεση κώδικα από άλλο εκτελέσιμο

Για την υλοποίηση του κώδικα αυτού του ερωτήματος ξεκινήσαμε διαβάζοντας το manual της exec και πιο συγκεκριμένα της execv. Αφού καταλάβαμε πως δουλεύει η execv ξεκινήσαμε γράφοντας ότι χρειάζεται να κληρονομήσει το παιδί πριν καλέσουμε την fork, τις μεταβλητές pid_t για να κρατήσει το PID του παιδιού και άλλες σχετικές πληροφορίες, μια μεταβλητή status που θα χρησιμοποιηθεί για να αποθηκεύσει την κατάσταση της αναμονής και ένα πίνακα χαρακτήρων args που περιέχει τα επιχειρήματα για το πρόγραμμα που θα εκτελεστεί από την διεργασία παιδί. Το πρώτο στοιχείο είναι το μονοπάτι προς το εκτελέσιμο του προγράμματος και τα υπόλοιπα είναι τα επιχειρήματα που λαμβάνει το argv από τη γραμμή εντολών τα οποία είναι αναγκαία για να λειτουργήσει το πρόγραμμα που θέλουμε να εκτελέσουμε. Αφού γίνει η fork και επιστραφεί η τιμή 0 δηλαδή ότι βρισκόμαστε στην διεργασία παιδί καλείται η execv με τα επιχειρήματα που είχαν οριστεί νωρίτερα για να εκτελέσει το πρόγραμμα ./a1.1 με τα δεδομένα από τα argv[1], argv[2], argv[3].

3 Διαδιεργασιακή Επικοινωνία

Ο κώδικας αυτού του ερωτήματος είναι παρόμοιος με αυτόν του ερωτήματος 2 δηλαδή το άνοιγμα και κλείσιμο των αρχείων γίνεται στον πατερά με μια διαφορά πως σε αυτή την περίπτωση το γράψιμο στο αρχείο εγγραφής γίνεται επίσης από τον πατερά. Έχουμε προσθέσει μια global μεταβλητή n στην αρχή του κώδικα, η οποία ορίζει πόσες διεργασίες θα δημιουργηθούν. Χρησιμοποιούμε την εντολή signal για να εντοπίσουμε το σήμα SIGINT που θέλουμε και όταν το κάνουμε αυτό καλούμε την συνάρτηση handler που έχουμε ορίσει στην αρχή του κώδικα η οποία διαχειρίζεται το σήμα, στην περίπτωση μας θα τυπώνει την σταθερά n η οποία εκφράζει τον αριθμό των διεργασιών οι οποίες αναζητούν τον χαρακτήρα. Μέσα σε ένα βρόγχο ο οποίος επαναλαμβάνεται n φορές δημιουργούμε ένα pipe και κάνουμε fork την γονική διεργασία, με αυτό τον τρόπο εξασφαλίζουμε ότι κάθε διεργασία παιδί που θα δημιουργείτε θα έχει και το δικό του pipe. Έπειτα, κλείνουμε το read end του pipe της διεργασίας παιδί, διότι το παιδί θέλουμε να γραφεί μόνο. Και ξαναχρησιμοποιούμε την εντολή signal αυτή τη φορά όχι για να διαχειριστούμε το σήμα που ψάχνουμε αλλά για να το αγνοήσουμε με τη χρήση του SIG_IGN. Στη συνέχεια εκτελούμε τη read, γράφουμε το αποτέλεσμα του counter στο pipe και βάζουμε το παιδί να κοιμηθεί για 1 δευτερόλεπτο ώστε να έχουμε χρόνο κατά την εκτέλεση να στείλουμε το σήμα SIGINT. Τέλος, κλείνουμε το write end του pipe από την πλευρά του πατερά, καλούμε την explain_wait_status για να μας εξηγήσει πως εκτελέστηκαν τα παιδιά και πηγαίνουμε και διαβαζουμε από το pipe το αποτέλεσμα που μας έστειλε το παιδί. Αυτό το κάνουμε μέσα σε ένα βρόγχο ο οποίος θα τρέξει n φορές δηλαδή για όλα τα παιδιά και στο τέλος χρησιμοποιούμε το count για να αθροίσουμε όλα τα counters δηλαδή τα αποτελέσματα όλων των παιδιών. Αφού το κάνουμε αυτό πηγαίνουμε και γράφουμε στο αρχείο εγγραφής το count και κλείνουμε τα ανοιχτά αρχεία.