Πανεπιστήμιο Δυτικής Μακεδονίας
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα

Ενότητα 9: Διεργασίες Zombie. Συναρτήσεις: pipe(), write(), read(), perror(), open(), socketpair(), socket(), listen(), accept(), send(), write(), recv().

Επιβλέπων: Δασυγένης Μηνάς
Παυλίδου Ελένη

Δρ. Μηνάς Δασυγένης
mdasyg@ieee.org
Εργαστήριο Λειτουργικών Συστημάτων
http://arch.ece.uowm.gr/courses/os/

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

zombie Διεργασίες (1/3)

- zombie διεργασία, λέμε μια διεργασία Α της οποίας ο γονέας δεν έχει αποδεχθεί τον κωδικό εξόδου της Α. Για όλο αυτό το διάστημα η διεργασία Α είναι μια ζωντανή-νεκρή (zombie) διεργασία.

- Οι zombie διεργασίες καταναλώνουν άσκοπα μνήμη (είναι εκεί παρόλο που έχουν ολοκληρώσει την λειτουργία τους).

zombie Διεργασίες (2/3)

- Π.Χ.:

```
#include<unistd.h>
int main()
{
 int pid;
 pid = fork();
if (pid == -1) { /*Check for error*/
  perror("fork");
  exit(1); }
 else if (pid == 0)
    { /* The child process */ exit(37);
      /* Exit with a silly number */ }
else { /* The parent process */ while (1) /*
  Never terminate */ sleep(1000); }
      }
```

Επειδή ο πατέρας δεν τερματίζει ποτέ, το παιδί δεν μπορεί να παραδώσει το exit(37).

Zombie Διεργασίες (3/3)

Εκτέλεση

```
$./zombie
& [1] 20842
$ ps -a
```

PID	TTY	TIME	CMD
6244	pts/3	00:00:00	su
6245	pts/3	00:00:00	bash
20842	pts/0	00:00:00	
20843	pts/0	00:00:00	zombie
20844	pts/0	00:00:00	zombie

`ps $kill -9 20842` Κάνοντας `kill` τον γονέα επιτρέπει στο παιδί να τερματίσει την εκτέλεσή του.

`$ps –a | grep zombie` Επομένως εδώ δεν επιστρέφεται τίποτα.

Συνάρτηση pipe() (1/3)

```
int pipe(int fd[2])
```

- Ανοίγει μια σωλήνωση και επιστρέφει δύο περιγραφείς αρχείου, `fd[0]`, `fd[1]`.
  Ο περιγραφέας `fd[0]` χρησιμοποιείται για ανάγνωση ενώ ο `fd[1]` για εγγραφή.

- Η συνάρτηση `pipe()` επιστρέφει `0` στην επιτυχία και `-1` στην αποτυχία, θέτοντας το `errno` στην κατάλληλη τιμή.

Συνάρτηση pipe() (2/3)

- Η ανάγνωση και εγγραφή γίνονται συνήθως με τη χρήση των συναρτήσεων `read()` και `write()`, αλλά μπορούν να χρησιμοποιηθούν και συναρτήσεις μορφοποιημένης Ε/Ε (τόσο δυαδικής όσο και κειμένου) εάν συνδέσουμε τους περιγραφείς αρχείου με ροές (δείκτη σε αρχείο) μέσω της συνάρτησης `fdopen()`.

- Μια σωλήνωση που ανοίγει με `pipe()` πρέπει πάντα να κλείνει με διπλή χρήση της συνάρτησης `close(int fd)`.

Συνάρτηση pipe() (3/3)

- Παράδειγμα: Η γονική διεργασία στέλνει δεδομένα στη θυγατρική διεργασία.

```
int pdes[2];
pipe(pdes);
if (fork() == 0)
{ /*this is the child process */
  close(pdes[1]);  /* it is not required, it will read */
   read(pdes[0]);  /* read from parent */ …
  }
else
{ /* this is the parent process */
close(pdes[0]);  /* it is not required, it will write */
write(pdes[1]);  /* write to child */…
}
```

Η συνάρτηση write() (1/2)

- Από τη στιγμή που ένα αρχείο έχει ανοιχτεί για γράψιμο μπορεί να προσπελαστεί με τη `write()`.
  Η δήλωσή της είναι:

```
int write (int fd, void  *buf, int size);
```

- Κάθε φορά που η `write()` εκτελείται, γράφονται `size` χαρακτήρες στο αρχείο το οποίο καθορίζεται από το `fd`, από τον `buffer *buf`.

Η συνάρτηση write() (2/2)

- Επειδή η `write()` μπορεί να γράψει έναν buffer ο οποίος είναι μισογεμάτος, δεν γράφονται όλα τα περιεχόμενα του buffer στο δίσκο. Η συνάρτηση επιστρέφει τον αριθμό των bytes που γράφτηκαν έπειτα από μια επιτυχή κλήση της, ενώ σε αποτυχία επιστρέφεται `–1`.

- Η συνάρτηση `read()` αποτελεί συμπλήρωμα της `write()`.

Η συνάρτηση read()

```
int read (int fd, void  *buf, int size);
```

όπου οι παράμετροι `fd`, `buf` και `size` αντιδρούν όπως και στην `write()` με τη διαφορά ότι η `read()` τοποθετεί δεδομένα στον buffer που δείχνει ο pointer `buf`. Αν η `read()` επιτύχει, επιστρέφει τον αριθμό χαρακτήρων που διαβάστηκαν, ενώ επιστρέφει `–1` σε περίπτωση σφάλματος.

Η συνάρτηση perror() (1/2)

- Η συνάρτηση `perror()` δηλώνεται ως εξής:

```
void perror(const char *message);
```

- Η `perror()` παράγει ένα μήνυμα (στο πρότυπο σφάλμα, `stderr`) που περιγράφει το τελευταίο σφάλμα που συνέβη κατά την κλήση μιας συνάρτησης βιβλιοθήκης ή κλήσης συστήματος και το οποίο επέστρεψε έναν αριθμό σφάλματος `errno`.

Η συνάρτηση perror() (2/2)

- Το όρισμα `message`, εμφανίζεται πρώτο, μετά ένα ερωτηματικό και ένα κενό. Κατόπιν ακολουθεί το μήνυμα του συστήματος που αντιστοιχεί στο `errno`.
  Αν το `message` είναι δείκτης NULL ή είναι κενό, το ερωτηματικό δεν εμφανίζεται, αλλά μόνο το μήνυμα του συστήματος.

- Επίσης υπάρχουν και ορισμένες άλλες συναρτήσεις εμφάνισης σφαλμάτων, όπως η `strerror()` η οποία αποθηκεύει τα μηνύματα σε απομονωτή που επιλέγει ο προγραμματιστής. Αυτές οι συναρτήσεις βρίσκονται στη `string.h`.

Η συνάρτηση open() (1/3)

- Άνοιγμα και δημιουργία αρχείων:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int oflag,
... /* mode_t mode */);
```

- Επιστρέφει περιγραφέα αρχείου αν επιτύχει, αλλιώς –1

Η συνάρτηση open() (2/3)

- Το πρώτο όρισμα υποδεικνύει τη διαδρομή του αρχείου που θέλουμε να ανοιχθεί.

- Το δεύτερο όρισμα υποδεικνύει τον τρόπο ανοίγματος του αρχείου.

  - Ορίζεται ως διάζευξη επιπέδου bit συμβολικών σταθερών.

- Το τρίτο όρισμα καθορίζει τα δικαιώματα πρόσβασης στο αρχείο, εφ’ όσον αυτό θα δημιουργηθεί.

Η συνάρτηση open() (3/3)

```
int open (char  *filename, int mode, int
access);
```

όπου `filename` είναι ένα οποιοδήποτε αποδεκτό όνομα αρχείου και `mode` ένα από τα παρακάτω macros δηλωμένο στο `<fcntl.h>`.

`O_RDONLY`	Read Only
`O_WRONLY`	Write Only
`O_RDWR`	Read / Write

Η συνάρτηση socket()

- Οι υποδοχές (*sockets*) παρέχουν σημειακή, αμφίδρομη επικοινωνία μεταξύ διεργασιών, συνήθως (αλλά όχι απαραίτητα), μέσω δικτύου. Αποτελούν μια πολύ ευέλικτη και θεμελιώδη μέθοδο δικτυακής επικοινωνίας μεταξύ διεργασιών ή συστημάτων.

- Η συνάρτηση `int socket(int domain, int type, int protocol)`, καλείται από μια διεργασία για τη δημιουργία μιας υποδοχής τύπου `type`, στο πεδίο `domain` και με πρωτόκολλο `protocol`.
  Αν το πρωτόκολλο δεν οριστεί (τιμή `0`), το σύστημα χρησιμοποιεί ένα προεπιλεγμένο πρωτόκολλο για το συγκεκριμένο τύπο. Η συνάρτηση επιστρέφει τον περιγραφέα της υποδοχής.

Η συνάρτηση socketpair() (1/4)

```
int socketpair(int af, int type, int
protocol, int sv[2])
```

- Η κλήση της `socketpair()` έχει σαν αποτέλεσμα τη δημιουργία δύο συνδεδεμένων sockets. `sv[]` είναι ο πίνακας όπου οι file descriptors για τα sockets αποθηκεύονται. Κάθε file descriptor στον `sv[]` συνδέεται με ένα άκρο της γραμμής επικοινωνίας. Κάθε file descriptor μπορεί να χρησιμοποιηθεί είτε για είσοδο είτε για έξοδο. Αυτό σημαίνει ότι είναι δυνατή η πλήρης διπλής-κατεύθυνσης επικοινωνία μεταξύ δύο διεργασιών (child process και parent process).

Η συνάρτηση socketpair() (2/4)

- Κανονικά, ένας file descriptor είναι δεσμευμένος προς χρήση από την parent process και ο άλλος file descriptor χρησιμοποιείται από την child process.
  Η parent process κλείνει τον file descriptor που χρησιμοποιείται από την child process.
  Αντιθέτως, η child process κλείνει τον file descriptor που χρησιμοποιείται από την parent process. Το `fork()` απαιτείται για να περάσει ένα socket σε μία child process.

Η συνάρτηση socketpair() (3/4)

- Το πρώτο όρισμα της `socketpair()`, το `af` ορίζει την οικογένεια στην οποία ανήκει το socket. Οι δύο οικογένειες που χρησιμοποιούνται αρκετά συχνά είναι:

- AF_UNIX για την επικοινωνία μεταξύ διεργασιών στην ίδια μηχανή.

- AF_INET για την επικοινωνία μεταξύ διεργασιών που βρίσκονται σε διαφορετικές μηχανές χρησιμοποιώντας τα τυποποιημένα πρωτόκολλα DARPA (IP/UDP/TCP).

Η συνάρτηση socketpair() (4/4)

- Το δεύτερο όρισμα, `type`, είναι ο τύπος του socket και έχει να κάνει με το «στυλ» της επικοινωνίας. Οι δύο συνηθέστερα χρησιμοποιημένες τιμές περιλαμβάνουν:

- SOCK_STREAM: Μια ροή δεδομένων χωρίς όρια μεγέθους. Η παράδοση σε ένα δικτυωμένο περιβάλλον είναι εγγυημένη. Εάν η παράδοση είναι αδύνατη, ο αποστολέας λαμβάνει ένα μήνυμα λάθους.

- SOCK_DGRAM: Μια ροή δεδομένων όπου κάθε πακέτο έχει συγκεκριμένο μέγεθος. Η παράδοση σε ένα δικτυωμένο περιβάλλον δεν είναι εγγυημένη.

Η συνάρτηση listen() (1/2)

```
int listen(int sd, int backlog);
```

- Η δεύτερη παράμετρος `backlog` προσδιορίζει το μήκος της ουράς συνδέσεων για τον socket descriptor `sd` και καθορίζει ότι θα δέχεται συνδέσεις. Το λειτουργικό σύστημα επιβάλλει ένα μέγιστο στο μήκος της ουράς συνδέσεων. Αν η παράμετρος του μήκους είναι μεγαλύτερη από αυτό, το μήκος της ουράς τίθεται χωρίς προειδοποίηση σε αυτό το μέγιστο.

Η συνάρτηση listen() (2/2)

- Ουσιαστικά η κλήση αρχικοποιεί την ουρά συνδέσεων και οποιεσδήποτε αιτήσεις συνδέσεων γίνουν (ενώ ο εξυπηρετητής είναι απασχολημένος) θα παραμένουν στην ουρά. Αν η ουρά γεμίσει τότε οι καινούριες αιτήσεις για σύνδεση θα χαθούν. Για παράδειγμα η παρακάτω κλήση:

```
listen(sd, num);
```

ορίζει το μήκος της ουράς σε `num` αναμένουσες κλήσεις σύνδεσης.

Η συνάρτηση accept() (1/4)

```
int accept(int sockfd, struct sockaddr *name
, int *namelen)
```

- Η κλήση της `accept()` εγκαθιστά μια σύνδεση πελάτη-εξυπηρετητή από την πλευρά του εξυπηρετητή. (Ο πελάτης ζητά τη χρησιμοποίηση της σύνδεσης με την κλήση της `connect()`). Ο εξυπηρετητής πρέπει να έχει δημιουργήσει το socket χρησιμοποιώντας την `socket()`, να έχει δώσει ένα όνομα στο socket με την `bind()`, και να την ενεργοποιήσει περιμένοντας νέες αιτήσεις με την `listen()`.

Η συνάρτηση accept() (2/4)

- Το `sockfd` είναι ο file descriptor για το socket ο οποίος επιστρέφεται από την `socket()` και `name` είναι ένας δείκτης σε μια δομή του τύπου sockaddr:

```
  struct sockaddr
{
  u_short sa_family;
  char sa_data[14];
}
```

Η συνάρτηση accept() (3/4)

- Μετά από μία επιτυχημένη κλήση της `accept()` αυτή η δομή θα περιέχει τη διεύθυνση πρωτοκόλλου του socket του πελάτη.
  Σε αυτή την περίπτωση η `accept()` δημιουργεί ένα νέο socket της ίδιας οικογένειας, τύπου, και πρωτοκόλλου όπως το sockfd. Ο file descriptor για αυτό το νέο socket είναι η τιμή που επιστρέφει η `accept()`.

Η συνάρτηση accept() (4/4)

- Αυτό το νέο socket χρησιμοποιείται για όλες τις επικοινωνίες με τον πελάτη.
  Εάν δεν υπάρχει καμία αίτηση στην ουρά αναμονής αιτήσεων η `accept()` θα μπλοκάρει μέχρι να παρουσιαστεί μια νέα αίτηση.
  Όταν παρουσιαστεί κάποιο πρόβλημα η `accept()` επιστρέφει `-1` και το errno αρχικοποιείται ανάλογα περιγράφοντας το λάθος.

Η συναρτήσεις send() και recv()

- Με μια ενεργή socket σύνδεση μπορούμε να στείλουμε δεδομένα χρησιμοποιώντας τη συνάρτηση `send` και να παραλάβουμε δεδομένα χρησιμοποιώντας την `recv`.

- Η συνάρτηση `send` δέχεται ως παράμετρο το μήκος των δεδομένων προς αποστολή. Αν αποστέλλεται κείμενο, το μήκος προκύπτει με τη συνάρτηση `strlen()`, ενώ για οτιδήποτε άλλο, το μήκος προκύπτει με τον τελεστή `sizeof`.

- πχ: `send(sock, "Test String", 12, 0);`

Τέλος Ενότητας

Πανεπιστήμιο Δυτικής Μακεδονίας