

1 Zadanie

W wirtualnym świecie działa sklep XXX. Jego oferta składa się ze skończonej ilości towarów i z każdym towarem w magazynie jest związana następująca informacja:

- NAZWA - niepusty ciąg znaków alfabetu łacinskiego bez spacji tzn. `[a-zA-Z]\+`,
- CENA - liczba dodatnia całkowita będąca ceną,
- ILOSC - ile jednostek towaru jest w magazynie, liczba dodatnia.

Przy uruchamianiu programu obsługującego XXX, dane o magazynie ładowane są z pliku tekstowego, którego nazwa jest przekazywana jako pierwszy i jedyny parametr programu realizującego zadania sklepu. Plik magazynu jest następującego formatu:

1. W pierwszej linii jest liczba towarów.
2. Każda kolejna jest postaci `<NAZWA CENA ILOSC>` i opisuje jeden towar w magazynie.

Przykład pliku magazynu:

```
3
Myszka 20 3
Kubek 21 1
Trabka 10 200
```

Zakładamy, że plik magazynu jest zawsze poprawny.

Klienci sklepu XXX łączą się do niego przez specjalne SkTerminale i wydając mu polecenia mogą zakupić towar w sklepie. W danej chwili może działać wiele SkTerminali.

SkTerminal jest procesem, który umożliwia pobieranie ze standardowego wejścia następujących poleceń:

- "oferta" - Klient chce otrzymać informacje o ofercie.
 1. SkTerminal wysyła do XXX prośbę o przesłanie oferty tzn. listy dostępnych towarów postaci `<NAZWA,CENA>,...`
 2. Po otrzymaniu wypisuje ją.
- "mysle N" - Klient chce zastanowić się przez N sekund.
 1. SkTerminal zasypia na N sekund, gdzie N jest liczbą całkowitą.
- "kup NAZWA" - Klient chce kupić 1 jednostkę towaru o nazwie NAZWA.
 1. SkTerminal wysyła prośbę do XXX o kupno towaru NAZWA.
 2. Sklep sprawdza czy towar jest w magazynie,
 - (a) Jeśli tak to transakcja kupna jest pomyślna i XXX aktualizuje swój magazyn, przysyłając informacje do SkTerminala, który informuje szczęśliwego klienta.
 - (b) Jeśli nie to SkTerminal wypisuje informacje, że w magazynie już nie ma takiego towaru.
- "koniec" - Klient odchodzi od terminala.
 1. SkTerminal informuje o dokonanych transakcjach.
 2. SkTerminal kończy działanie.

Przykład interakcji klienta z skterminalem:

```
oferta
Myszka 20
Kubek 21
Trabka 10
mysle 2
kup Kubek
Zakupiony!
```

```

oferta
  Myszka 20
  Trabka 10
koniec
Kupiles 1 Kubek za 21

```

Zakładamy, że klienci są bardzo bogaci i mają mnóstwo pieniędzy.

Zadanie

Napisz programy `sklep.c` realizujący zadania XXX i `skterminal.c` realizujący zadania SkTerminala. Programy muszą korzystać z kolejki komunikatów do przesyłania informacji.

Proces w czasie działania wypisuje czytelnie informacje o tym co robi, aktualny stan magazynu i stan kasy.

2 Krótki opis działania

Sklep po uruchomieniu tworzy kolejkę komunikatów `/tmp/sklep_tri10o`. Jednocześnie, jeśli jest już taki plik, program uznaje, że w systemie działa inny sklep i kończy działanie. Również po uruchomieniu program ładuje stan magazynu i ustawia kasę na 0. Informacje o tym są wysyłane na konsolę. W dalszej części sklep przechodzi do pętli realizującej kolejne zapytania z kolejki. Zakończenie sklepu odbywa się poprzez wysłanie `SIGINT`. Sygnał ten powoduje wywołanie funkcji `bezpieczne_wyjście()` zamykającej sklep z jednoczesnym usunięciem kolejki, zamknięciem łącz i zwolnieniem pamięci (magazynu).

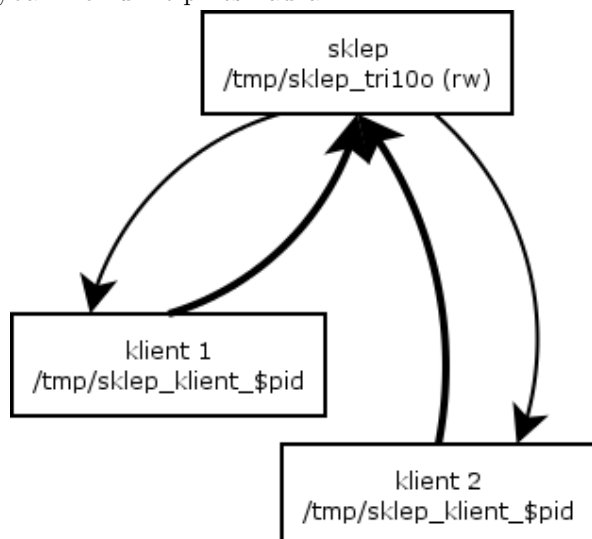
Większość wydarzeń w sklepie jest logowana (`logowanie()`). Zawsze są rejestrowane operacje związane z kasą (funkcje `kasa()`, `nabij_na_kase()`) i polecenia klientów.

Klient (`skterminal`) po uruchomieniu tworzy własną kolejkę, na którą będzie odbierał informacje ze sklepu, otwiera kolejkę sklepu (`/tmp/sklep_tri10o`) i rejestruje się jako nowy. Klient składa się z dwóch procesów. Głównego, przetwarzającego polecenia użytkownika na komunikaty dla sklepu i drugiego, wyświetlającego bajt po bajcie wszystko, co spłynie z serwera na kolejkę `skterminala`. A więc sklep wysyła wszystkie odpowiedzi w postaci tekstu, czytelnego dla człowieka. Sklep kończy życie z poleceniem `koniec`, usuwając swoją kolejkę i sygnalizując to sklepowi.

3 Komunikacja

Sklep przyjmuje komunikaty typu `komunikat_t`, zawierające pole nadawcy i rodzaju operacji. Jeśli komunikat mówi o utworzeniu nowego klienta, lub o chęci zakupu towaru, przesyłany jest jeszcze argument, czyli albo nazwa kolejki `fifo` klienta, albo nazwa towaru do kupienia.

Jeśli klient został utworzony, serwer otwiera jego `fifo` i w odpowiedzi przesyła jego deskryptor, nie zapamiętując nigdzie. Klient w przyszłości, aby dostać jakąś odpowiedź, musi przesyłać numer deskryptora. W prosty sposób jest więc możliwe kupowanie na konto innego klienta, ale w sytuacji gdy wszyscy klienci mają nieskończenie wiele pieniędzy, to chyba nikomu nie przeszkadza.



4 Kod źródłowy

4.1 struktury.h

```

#define SCIEZKA_LACZA "/tmp/sklep_tri10o" /* sciezka lacza do sklepu */
#define SCIEZKA_KLIENTA "/tmp/sklep_klient_" /* sciezka lacza do konkretnego klienta zwiekszona o numer PL

typedef struct towar {
    char *nazwa;
    unsigned int cena;
    unsigned int ilosc;
} towar_t;

#define OP_NOOP          0 /* brak operacji */
#define OP_NOWE_POL      1 /* otworzyc nowe lacze, ktorego nazwe przesyła klient */
#define OP_KONIEC_POL    2 /* zamknac lacze danego klienta, usunac go z listy klientow */
#define OP_OFERTA        4 /* zapytanie o oferte */
#define OP_KUP           8 /* chec kupienia towaru podanego w argumentcie */

/* struktura przesyłanych komunikatow.
   nadawca    - ktory klient nadaje. zeby wiedziec, na ktore lacze odpowiedziec.
   operacja    - jedna ze stalych OP_

   drugie slowo komunikacyjne dodatkowo: (tylko dla OP_KUP i OP_NOWE_POL)
   dlugosc     - dlugosc dodatkowego argumentu ciag_arg (potrzebny w OP_KUP i OP_NOWE_POL)
   ciag        - dla OP_NOWE_POL zawiera nazwe pliku lacza, dla OP_KUP nazwe towaru
*/
typedef struct komunikat {
    int nadawca;
    char operacja;
} komunikat_t;

typedef struct komunikat_arg {
    char dlugosc;
    char *ciag;
} komunikat_arg_t;

int otworz_lacze(char *sciezka, int flagi) {
    int lacze;
    if (mkfifo(sciezka, 0666) != 0)
        return 0;
    if ((lacze = open(sciezka, flagi)) == -1)
        return 0;
    return lacze;
}

```

4.2 sklep.c

```

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <time.h>

```

```

#include <signal.h>
#include <stdarg.h>

#include "struktury.h"

#define BUFSIZE 1024

/* sposob wykonania funkcji:
   - nie wykonywac, tylko zapamietac z jakimi zostala wywolana argumentami;
   - wykonac z takimi argumentami, z jakimi zostala wywolana ostatnio;
   - wykonac z takimi argumentami, jakie podano w biezacym wywołaniu;
*/
typedef enum tryb {zapamietaj_ustawienia, wykonaj, wykonaj_dla_argumentow} tryb_t;
typedef enum operacja_kasy {KASA_STAN='=', KASA_KP='+', KASA_KW='-'} operacja_kasy_t;

void logowanie(char *format, ...) {
    va_list ap;
    time_t teraz_sec = time(NULL);
    struct tm teraz;
    va_start(ap, format);
    gmtime_r(&teraz_sec, &teraz);
    fprintf(stderr, "sklep [%d-%02d-%02d %02d:%02d:%02d] : ", 1900+teraz.tm_year, teraz.tm_mon,
                                                         teraz.tm_mday, teraz.tm_hour,
                                                         teraz.tm_min, teraz.tm_sec);

    vfprintf(stderr, format, ap);
    va_end(ap);
}

int kasa(int kwota, operacja_kasy_t operacja) {
    static int stan_kasy=0;
    switch (operacja) {
        case (KASA_KP):
            stan_kasy+=kwota;
            break;
        case (KASA_KW):
            stan_kasy-=kwota;
            break;
    }
    logowanie("Stan kasy: %d (%c)\n", stan_kasy, operacja);
    return stan_kasy;
}

void nabij_na_kase(towar_t towar) {
    logowanie("Zakup \"%s\" za: %d. Na magazynie: %d\n", towar.nazwa, towar.cena, towar.ilosc);
    kasa(towar.cena, KASA_KP);
}

towar_t* laduj_towar(char *plik, int *l_towarow) {
    char buf[BUFSIZE]={0};
    int towarow_do_zaladowania=0, nr_towaru=0;
    towar_t* towary=NULL;
    FILE *p;
    if ((p = fopen(plik, "r")) == NULL)
        return NULL;
    fgets(buf, BUFSIZE, p);
    towarow_do_zaladowania = *l_towarow = atoi(buf);
    if ((towary = (towar_t*) malloc(towarow_do_zaladowania*sizeof(towar_t))) == NULL)

```

```

    return NULL;
while (towarow_do_zaladowania-- > 0) {
    fgets(buf, BUFSIZE, p);
    towary[nr_towaru].nazwa = (char*) malloc((int)memchr(buf, ' ', BUFSIZE)-(int)buf+1);
    if (towary[nr_towaru].nazwa==NULL) return NULL;
    /* dlugosc ciagu dla pola nazwa = adres bufora - adres pierwszej spacji w buforze + 1bajt na \0 */
    sscanf(buf, "%s %d %d", towary[nr_towaru].nazwa, &towary[nr_towaru].cena, &towary[nr_towaru].ilosc);
    nr_towaru++;
}
return towary;
}

void wypisz_towary(int lacze, towar_t* towar, int l_towarow) {
    char buf[23]={0};
    int k;
    for (k=0; k<l_towarow; k++) {
        sprintf(buf, "%-14s %5d\n", towar[k].nazwa, towar[k].cena);
        /* uwaga, dlugosc nie wieksza, niz rozmiar buf */
        write(lacze, buf, strlen(buf));
    }
}

int sprzedaj_towar(char *nazwa, towar_t* magazyn, int l_towarow) {
    int i, dl_naz, dl;
    dl_naz=strlen(nazwa);
    for (i=0; i<l_towarow; i++)
        if ((strncmp(nazwa, magazyn[i].nazwa, dl_naz<strlen(magazyn[i].nazwa) ? dl_naz : strlen(magazyn[i].nazwa))==0)
            (magazyn[i].ilosc>0)) {
            magazyn[i].ilosc--;
            nabij_na_kase(magazyn[i]);
            return 1;
        }
    return 0;
}

int dodaj_klienta(char* nowy) {
    int id;
    if ((id = open(nowy, O_WRONLY)) == -1)
        return 0;
    else
        return id;
}

void oferuj(int lacze, towar_t* towar, int l_towarow) {
    komunikat_t zapytanie;
    komunikat_arg_t argument={0, NULL};
    char argument_const[100]={0};
    char tmp;
    int k;
    while (read(lacze, &zapytanie, sizeof(komunikat_t))) {
        if (zapytanie.operacja & (OP_NOWE_POL|OP_KUP) ) { /* pobierz argument */
            read(lacze, &argument.dlugosc, sizeof(char));
            memset(argument_const, 0, 100);
            read(lacze, argument_const, argument.dlugosc);
        }

        switch (zapytanie.operacja) {

```

```

case (OP_NOWE_POL):
    if (k = dodaj_klienta(argument_const)) {
        write(k, &k, sizeof(int)); /* odpowiedzia jest numer przyznanego lacza,
                                   sklep nie zapamietuje listy otwartych lacz. */
        logowanie("Nowe połączenie: Klient %d\n", k);
    } else {
        logowanie("Błąd nowego połączenia\n");
    }
    break;
case (OP_KONIEC_POL):
    if (close(zapytanie.nadawca)==0)
        logowanie("Klient %d koniec połączenia: Sukces\n", zapytanie.nadawca);
    else
        logowanie("Klient %d koniec połączenia: Błąd\n", zapytanie.nadawca);
    break;
case (OP_OFERTA):
    logowanie("Klient %d oferta\n", zapytanie.nadawca);
    wypisz_towary(zapytanie.nadawca, towar, l_towarow);
    break;
case (OP_KUP):
    if (sprzedaj_towar(argument_const, towar, l_towarow)) {
        write(zapytanie.nadawca, "Towar zakupiony\n", 16);
        logowanie("Klient %d zakup dokonany\n", zapytanie.nadawca);
    } else {
        write(zapytanie.nadawca, "Brak towaru/Towar nieznany\n", 27);
        logowanie("Klient %d zakup nie dokonany\n", zapytanie.nadawca);
    }
    break;
}
}
}

/* gdy tryb==zapamietaj ustawienia, nie zamyka sklepu, tylko
   zapamietuje co bedzie trzeba zamknac w razie zamykania z bezpieczne_wyjście(),
   gdzie nie mozna przekazac parametrow, bo to jest signal handler */
void zamknij_sklep(int lacze, towar_t* towar, int ltow, tryb_t tryb) {
    static int ost_lacze, ost_ltow;
    static towar_t* ost_towar;
    int i;
    if (tryb & (zapamietaj_ustawienia|wykonaj_dla_argumentow)) {
        ost_lacze=lacze;
        ost_towar=towar;
        ost_ltow=ltow;
    }
    if (tryb & (wykonaj|wykonaj_dla_argumentow)) {
        close(ost_lacze);
        unlink(SCIEZKA_LACZA);
        for (i=0; i<ost_ltow; i++)
            free(ost_towar[i].nazwa);
        free(ost_towar);
    }
}

void bezpieczne_wyjście(int signum) {
    logowanie("Otrzymano sygnał wyjścia\n");
    zamknij_sklep(0, NULL, 0, wykonaj); /* */
    exit(0);
}

```

```

}

int main(int argc, char *argv[]) {
    towar_t* magazyn=NULL;
    int lacze, ltow;
    if (argc<2) {
        fprintf(stderr, "Używanie: sklep plik\n"
            "\tplik - plik z zawartością magazynu\n");
        return 1;
    }
    if ((magazyn = laduj_towar(argv[1], &ltow)) == NULL) {
        fprintf(stderr, "Nie można załadować magazynu\n");
        return 2;
    }
    logowanie("Towar załadowany\n");
    if (! (lacze = otworz_lacze(SCIEZKA_LACZA, O_RDWR))) {
        fprintf(stderr, "Nie można utworzyć łącza. Może działa już inny sklep, "
            "lub ostatnio program zamknięto niepoprawnie.\n");
        return 3;
    }
    kasa(0, KASA_STAN);
    logowanie("Połączenie otwarte\n");
    signal(SIGINT, bezpieczne_wyjscie);

    oferuj(lacze, magazyn, ltow);
    zamknij_sklep(lacze, magazyn, ltow, wykonaj_dla_argumentow);
    return 0;
}

```

4.3 skterminal.c

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include "struktury.h"

char *sciezka_klienta(void) {
    char* sciezka;
    sciezka = (char*) malloc(strlen(SCIEZKA_KLIENTA)+5); /* piec znakow na pid */
    sprintf(sciezka, SCIEZKA_KLIENTA"%d", getpid());
    return sciezka;
}

int polacz(int *od_serwera, int *do_serwera) {
    int id;
    komunikat_t powitanie={0, OP_NOWE_POL};
    komunikat_arg_t powitanie_cd;
    powitanie_cd.ciag=sciezka_klienta();
    powitanie_cd.dlugosc=strlen(powitanie_cd.ciag);
    if ((*do_serwera = open(SCIEZKA_LACZA, O_WRONLY)) == -1)
        return 0;
    if (! (*od_serwera = otworz_lacze(powitanie_cd.ciag, O_RDWR)))
        return 0;
    write(*do_serwera, &powitanie, sizeof(komunikat_t));
    write(*do_serwera, &powitanie_cd.dlugosc, 1);
}

```

```

write(*do_serwera, powitanie_cd.ciag, powitanie_cd.dlugosc);
read(*od_serwera, &id, sizeof(int));
return id;
}

void na_ekran(int serwer) {
    char znak;
    while (read(serwer, &znak, 1)) {
        putchar(znak);
    }
}

int zakupy(int do_serw, int id) {
    char buf[32];
    char kontynuluj=1;
    int czas=0;
    komunikat_t zapytanie={id, OP_NOOP};
    komunikat_arg_t argument={0, NULL};

    while (kontynuluj && fgets(buf, 32, stdin)) {
        if (strncmp("kup ", buf, 4) ==0 ) {
            zapytanie.operacja=OP_KUP;
            argument.dlugosc=strlen(buf)-5;
            argument.ciag = (char*) malloc(argument.dlugosc);
            memcpy(argument.ciag, buf+4, argument.dlugosc); /* usun "kup " z początku i \n z końca */
        }
        else if (strncmp("oferta\n", buf, 7) ==0 ) {
            zapytanie.operacja=OP_OFERTA;
        }
        else if (strncmp("mysle ", buf, 6) ==0 ) {
            if ((czas=atoi(buf+6))>0) {
                printf("Przerwa na myślenie: %d sekund.\n", czas);
                sleep(atoi(buf+6));
                printf("Koniec myślenia\n");
            } else {
                printf("mysle N (N czas w sekundach)\n");
            }
        }
        else if (strncmp("koniec\n", buf, 7) ==0 ) {
            zapytanie.operacja=OP_KONIEC_POL;
            kontynuluj=0;
        } else {
            printf("Nieznane polecenie. Polecenia:\n"
                "oferta\t\t- by poznać ofertę sklepu\n"
                "mysle N\t\t- jeśli chcesz chwilę pomyśleć\n"
                "kup NAZWA\t- aby dokonać zakupu\n"
                "koniec\t\t- aby zakończyć zakupy\n");
            fflush(stdout);
        }
        write(do_serw, &zapytanie, sizeof(komunikat_t));
        if (zapytanie.operacja == OP_KUP) {
            write(do_serw, &argument.dlugosc, 1);
            write(do_serw, argument.ciag, argument.dlugosc);
            argument.dlugosc=0;
            free(argument.ciag);
        }
    }
}

```



```
}

int main(void) {
    int do_serw, od_serw, id;
    if (! (id=polacz(&od_serw, &do_serw))) {
        fprintf(stderr, "Nie można wejść do sklepu. Może nie jest włączony?\n");
    }

    if (fork()==0) {
        /* proces odpowiedzialny za wyswietlanie */
        na_ekran(od_serw);
    } else {
        /* proces odpowiedzialny za wysylanie zapytan do serwera (glowny) */
        zakupy(do_serw, id);
        unlink(sciezka_klienta());
    }
    return 0;
}
```

90

100

5 Pliki

Kod źródłowy, program i wyniki działania znajdują się także pod adresem: <http://tri10o.republika.pl/so.shtml>