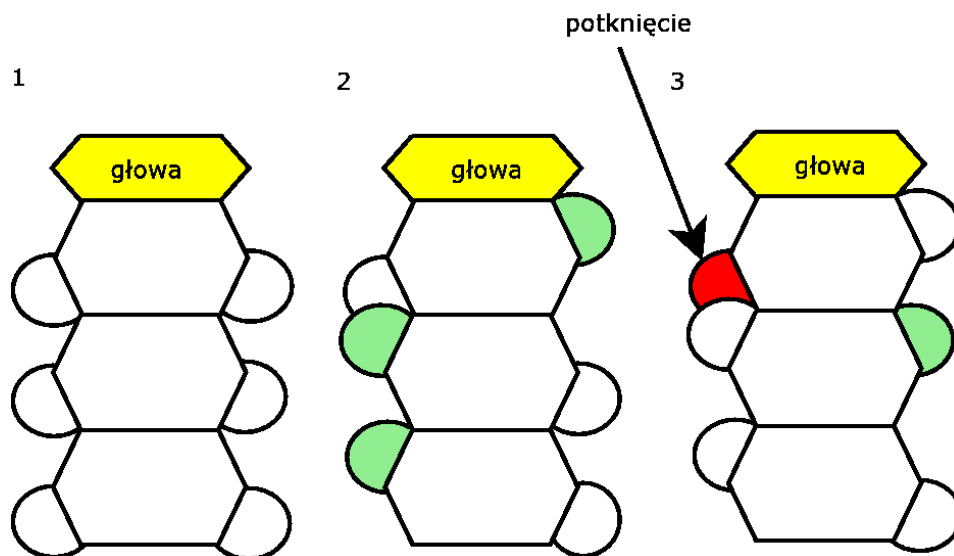


# 1 Stonoga

Stonoga składa się z głowy, pewnej liczby członów i nóg - dwóch dla każdego członu. Największy problem stonodze sprawia poruszanie się. Zadanie polega na zasymulowaniu chodzenia stonogi za pomocą procesów komunikujących się przy użyciu nienazwanych łączy komunikacyjnych (pipe'ów).

Program zostaje wywołany z jednym parametrem  $n$  - liczbą członów. Program składa się z następujących procesów: 1 głowy,  $n$  członów i  $2*n$  nóg (po dwie nogi dla każdego członu). Program symuluje ciąg kroków stonogi, każdy krok wygląda następująco:

- użytkownik daje głowie polecenie wykonania kroku, a ta przekazuje je pierwszemu członowi;
- każdy człon, gdy dostanie polecenie wykonania kroku, przekazuje je swoim nogom (lewej i prawej) i kolejnemu członowi (jeśli nie jest ostatnim);
- noga po otrzymaniu polecenia zrobienia kroku odczekuje pewien losowy czas, wykonuje krok i informuje o tym swój człon;
- gdy człon dostanie informację o wykonanym kroku od swojej nogi (którejkolwiek), to przekazuje ją poprzedniemu członowi. Jeśli jednak noga z następnego członu (tego z tyłu) wykona krok wcześniej niż odpowiadająca jej noga z bieżącego członu (następna lewa przed lewą lub następna prawa przed prawą), to stonoga potyka się (na standartowe wyjście wypisuje komunikat: lewa/prawa noga z członu nr. xxx potknęła się) w przeciwnym przypadku wypisuje: lewa/prawa noga z członu nr. xxx wykonała krok pomyślnie (tylko wypisanym komunikatem różni się potknięcie od poprawnego kroku);



- gdy głowa otrzyma informację o wykonanym kroku przez obie nogi pierwszego członu, to oczekuje na kolejne polecenie użytkownika - zrobienia kolejnego kroku albo zakończenia programu.

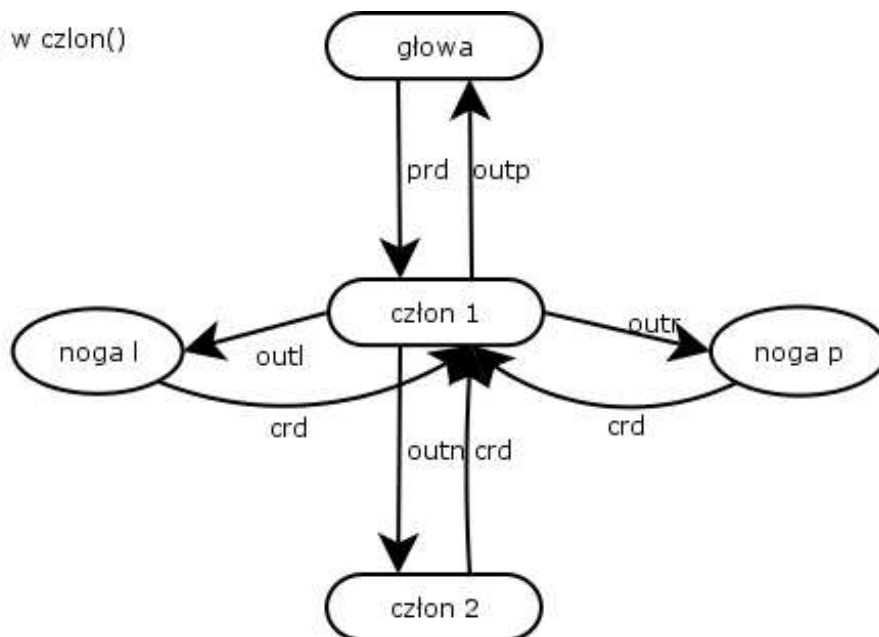
Procesy komunikują się za pomocą łączy nienazwanych.

## 2 Komunikacja

Procesy porozumiewają się przysyłając sobie wiadomości typu `t_msg`. Rozumiane wiadomości:

sender	content	opis
head	*	człon: przesłać swym nogom i następnemu członowi
left,right	step_done	człon: przekazać następnemu członowi w stronę głowy, że wykonano krok lewą/prawą nogą. Jeśli poprzedni człon już wysłał taką wiadomość, potknięcie.
prev	step_done	człon: zanotować, że poprzedni człon już wykonał krok lewą/prawą nogą.
	make_step	noga: wykonać krok
	call_exit	zakończyć działanie

łącza w czlon()



### 3 Działanie

```

bash:~/stonoga$ ./stonoga
stonoga n_czlonow
bash:~/stonoga$ ./stonoga 4
polecenia:
k - krok
q - wyjście
k
człon 2: prawy krok
człon 3: lewy krok
człon 4: prawy krok
człon 4: lewy krok
człon 2: lewy potknięcie
człon 1: prawy potknięcie
człon 3: prawy potknięcie
człon 1: lewy potknięcie
k
człon 2: prawy krok
człon 3: prawy krok
człon 3: lewy krok
człon 4: prawy krok
człon 4: lewy krok
człon 1: lewy krok
człon 1: prawy potknięcie
człon 2: lewy potknięcie
k
człon 1: prawy krok
człon 4: prawy krok
człon 2: lewy krok
człon 3: lewy krok
człon 2: prawy krok
człon 1: lewy potknięcie
człon 3: prawy potknięcie
człon 4: lewy krok
  
```

```

q
bash:~/stonoga$ ./stonoga 8
polecenia:
k - krok
q - wyjście
k
czlon 1: prawy krok
czlon 5: prawy krok
czlon 4: prawy potknięcie
czlon 7: prawy krok
czlon 2: lewy krok
czlon 7: lewy krok
czlon 3: prawy potknięcie
czlon 4: lewy krok
czlon 8: prawy krok
czlon 6: lewy potknięcie
czlon 1: lewy potknięcie
czlon 8: lewy krok
czlon 2: prawy potknięcie
czlon 5: lewy potknięcie
czlon 3: lewy potknięcie
czlon 6: prawy potknięcie
q
bash:~/stonoga$

```

## 4 Kod źródłowy

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/select.h>

```

```
typedef int t_pipe[2];
```

```
/* struktura wiadomosci */
```

```
enum t_sender {current=0, left, right, prev, head};
```

10

```
enum t_content {no_operation = 0,
                make_step    = 1,
                step_done    = 2,
                step_fail    = 4,
                left_step    = 8,
                right_step   = 16,
                call_exit    = 32};
```

```
struct msg {
    enum t_sender sender;
    enum t_content content;
};
```

20

```
typedef struct msg t_msg;
```

```
#define MSGSIZE sizeof(t_msg)
```

```
/* noga, leg - prawa, albo lewa, rd i wr - lacza do wlasciwego czlonu */
void noga(enum t_sender leg, int rd, int wr) {
```

```

t_msg msg={current, no_operation};
srand(getpid()*getppid()*time(NULL));
while (msg.content!=call_exit) {
    if (msg.content==make_step) {
        usleep((int)(3000000.0*rand()/(RAND_MAX+1.0)));
        msg.sender=leg;
        msg.content=step_done|((leg==left) ? left_step : right_step);
        write(wr, &msg, MSGSIZE);
    }
    read(rd, &msg, MSGSIZE);
}
close(rd); close(wr);
exit(0);
}

/* czlon n, czyta lacza prd i crd, zapisuje do out* */
void czlon(int n, int prd, int crd, int outl, int outr, int outn, int outp) {
    t_msg msg={current, no_operation};
    int s, i, left=0, right=0;
    fd_set set;
    struct timeval t;
    FD_ZERO(&set);

    while (msg.content!=call_exit) {
        do { /* czekaj, az select stwierdzi ze jest wiadomosc w prd lub crd */
            FD_SET(prd, &set);
            FD_SET(crd, &set);
            t.tv_sec = 1;
            t.tv_usec = 0;
        } while (select(crd+1, &set, NULL, NULL, &t)!=1);
        if (FD_ISSET(crd, &set)) s=read(crd, &msg, MSGSIZE);
        else s=read(prd, &msg, MSGSIZE);

        /* przetworzenie wiadomosci */
        if (msg.sender==head) { /* wiadomosc od glowy, przekaz nastepnemu i nogom */
            left=right=0; /* zresetuj stan nog poprzednich */
            write(outl, &msg, MSGSIZE);
            write(outr, &msg, MSGSIZE);
            write(outn, &msg, MSGSIZE);
        }

        if (msg.content&step_done&&msg.sender!=prev) { /* potwierdzenie kroku od nogi */
            msg.sender=prev; /* przygotowanie potwierdzenia dla poprzedniego czlonu */
            if ((msg.content&left_step && left==0)||
                (msg.content&right_step && right==0)) {
                printf("czlon %d: %s krok\n", n, msg.content&left_step ? "lewy" : "prawy");
            } else {
                printf("czlon %d: %s potknięcie\n", n, msg.content&left_step ? "lewy" : "prawy");
                msg.content^=step_done|step_fail; /* wykasuj step_done, ustaw step_fail */
            }
            write(outp, &msg, MSGSIZE); /* przekaz poprzedniemu */
        }

        if (msg.sender==prev) { /* krok poprzedniego czlonu wykonany */
            if (msg.content&left_step) left++;
            else right++;
        }
    }
}

```

```

    }
    close(crd); close(prd); close(outl); close(outp); close(outn); close(outp);
    exit(0);
}
90

/* glowa (wyk. w procesie glownym), rd i wr - lacza do pierwszego czlonu */
void glowa(int rd, int wr) {
    char c;
    t_msg msg={head, make_step};
    printf("polecenia:\nk - krok\nq - wyjscie\n");
    while ((c=getchar())!='q') {
        if (c=='k') write(wr, &msg, MSGSIZE);
    }
    msg.content=call_exit;
    write(wr, &msg, MSGSIZE);
    close(rd); close(wr);
}
100

/* utworzenie lacz. prev_rd i prev_wr - lacza do poprzedniego czlonu do read i write */
/* n - liczba laczy do utworzenia, cn - numer aktualnego czlonu */
void dodaj_czlony(int n, int cn, int prev_rd, int prev_wr) {
    int i;
    t_pipe curr_out[3];
    t_pipe curr_in;
    for (i=0; i<3; i++) pipe(curr_out[i]);
    pipe(curr_in);
    if (fork()==0) {
        close(curr_out[0][1]);
        close(curr_in[0]);
        noga(left, curr_out[0][0], curr_in[1]);
    }
    if (fork()==0) {
        close(curr_out[1][1]);
        close(curr_in[0]);
        noga(right, curr_out[1][0], curr_in[1]);
    }
    if (n>1)
        if (fork()==0) {
            close(curr_out[2][1]);
            close(curr_in[0]);
            dodaj_czlony(n-1, ++cn, curr_out[2][0], curr_in[1]);
            exit(0);
        }
    close(curr_in[1]);
    for (i=0; i<3; i++)
        close(curr_out[i][0]);
    czlon(cn, prev_rd, curr_in[0], curr_out[0][1], curr_out[1][1], curr_out[2][1], prev_wr);
}
110
120
130

int main(int argc, char *argv[]) {
    int n;
    t_pipe rd, wr;
    if (argc<2) {
        printf("stonoga n_czlonow\n"); return 1;
    }
    n=atoi(argv[1]);
    pipe(rd);
140

```

```
pipe(wr);
if (fork()==0) {
    close(rd[1]);
    close(wr[0]);
    dodaj_czlony(n, 1, rd[0], wr[1]);
}
close(rd[0]);
close(wr[1]);
glowa(wr[0], rd[1]);
return 0;
}
```

150