|  |
| --- |
| #include <sys/types.h> |
|  | #include <sys/ipc.h> |
|  | #include <sys/msg.h> |
|  | #include <pthread.h> |
|  | #include <stdio.h> |
|  | #include <string.h> |
|  | #include <stdlib.h> |
|  |  |
|  | struct Strmsg// Для того чтобы отправить или получить сообщения, вызывающий процесс создает структуру по такой общей форме: |
|  | { |
|  | long mtype;//тип сообщения |
|  | int digits[4]; |
|  | }; |
|  |  |
|  | struct args |
|  | { |
|  | int msgId; |
|  | }; |
|  |  |
|  | int sort(const void \*x, const void \*y)//разница между |
|  | { |
|  | return \*(int \*)x - \*(int \*)y; |
|  | } |
|  |  |
|  | void swap(int \*digits, int x, int y)//обмен местами |
|  | { |
|  | int tmp = digits[x]; |
|  | digits[x] = digits[y]; |
|  | digits[y] = tmp; |
|  | } |
|  |  |
|  | int gen(int \*digits, size\_t n)//генерируются все возможные перестановки 1234 |
|  | { |
|  | int j = n - 2; |
|  | while (j != -1 && digits[j] >= digits[j + 1]) |
|  | { |
|  | j--; |
|  | } |
|  | if (j == -1) |
|  | { |
|  | return 0; |
|  | } |
|  | int k = n - 1; |
|  | while (digits[j] >= digits[k]) |
|  | { |
|  | k--; |
|  | } |
|  | swap(digits, j, k); |
|  | int l = j + 1, r = n - 1; |
|  | while (l < r) |
|  | { |
|  | swap(digits, l++, r--); |
|  | } |
|  | return 1; |
|  | } |
|  |  |
|  | void \*mainThd(void \*thd\_args)//генерация массива из 4х чисел |
|  | { |
|  | srand(time(NULL));//это делает основной поток |
|  | int digits[4]; |
|  | for (size\_t i = 0; i < 4; ++i) |
|  | { |
|  | digits[i] = rand() % 100; |
|  | printf("%d ", digits[i]);//вывод на экран |
|  | } |
|  | printf("\n"); |
|  | // msgsnd() -- для передачи и msgrcv() -- для приема. |
|  | struct args \*args\_var = (struct args \*)thd\_args; |
|  | int msgId = args\_var->msgId;//передается идент сообщений |
|  | struct Strmsg message; |
|  | memcpy(message.digits, digits, 4 \* sizeof(int)); |
|  | message.mtype = 1;//готовим сообщение |
|  | msgsnd(msgId, &message, sizeof(message), 0); Первый  дескриптор очереди (таким образом, msgsnd() может обращаться только к существующей очереди). Второй -- указатель на сообщение – это структура, которая хранит собственно сообщение и его тип. |
|  |  |
|  | int count = 0; |
|  | do |
|  | { |
|  | size\_t size = msgrcv(msgId, &message, sizeof(message), 101, 0);// извлекает сообщения из очереди сообщений |
|  | if (message.digits[0] != -1) |
|  | { |
|  | for (size\_t i = 0; i < 4; ++i) |
|  | { |
|  | printf("%d ", message.digits[i]);//выводим на экран с дальнейшим продолжением |
|  | } |
|  | count ++; |
|  | printf("\n"); |
|  | } |
|  | } while (message.digits[0] != -1); |
|  | printf("count: %d\n", count);//количество перестановок |
|  |  |
|  | msgctl(msgId, IPC\_RMID, NULL); удаляется очередь |
|  | return NULL; |
|  | } |
|  |  |
|  | void \*childThd(void \*thd\_args)//для второго потока |
|  | { |
|  | struct args \*args = (struct args \*)thd\_args; |
|  | int msgId = args->msgId; |
|  | struct Strmsg message; |
|  | Здесь происходит перестановки…после передачи во второй процесс после выввода на на экран массива первым процессов. |
|  | msgrcv(msgId, &message, sizeof(message), 1, 0);//добавляет 1 параметр, это откуда пришло сообщение |
|  | qsort(message.digits, 4, sizeof(int), sort); |
|  |  |
|  | message.mtype = 101; |
|  | msgsnd(msgId, &message, sizeof(message), 0);//доб сообщение |
|  | while (gen(message.digits, 4)) |
|  | { |
|  | message.mtype = 101; |
|  | msgsnd(msgId, &message, sizeof(message), 0);//доб сообще |
|  | } |
|  | message.digits[0] = -1;//подготавливаем сообщения |
|  | message.mtype = 101; |
|  | msgsnd(msgId, &message, sizeof(message), 0);//отправить сообщение  первому процессу обратно и он уже принимает и выводит возможные комбинации и их количество и очищает очередь сообщений. |
|  | return NULL; |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | int msgId = msgget(IPC\_PRIVATE, 0600 | IPC\_CREAT);//созд разд памяти  if (msgId < 0) {  perror("error with msgget()");  return -1; }  else {  printf("message id = %i\n", msgId); } |
|  | struct args \*args\_var; |
|  | args\_var->msgId = msgId; |
|  |  |
|  | pthread\_t mainThread, childThread;//процесс родитель и ребенок |
|  |  |
|  | pthread\_create(&mainThread, NULL, mainThd, (void \*)args\_var);//1й поток |
|  | pthread\_create(&childThread, NULL, childThd, (void \*)args\_var);//2й поток |
|  |  |
|  | pthread\_join(mainThread, NULL); ожидает завершения потока первого, |
|  | } |