|  |  |  |
| --- | --- | --- |
| |  | | --- | | **INSTYTUT TELEINFORMATYKI I AUTOMATYKI**  **Wydział Cybernetyki WAT** | | |
| |  | | --- | | Przedmiot: SYSTEMY OPERACYJNE  SPRAWOZDANIE Z ĆWICZENIA LABORATORYJNEGO NR 1 | | |
| **Temat**: Kolejki komunikatów | |
| **Wykonał:**  xxxxx | **Data wykonania ćwiczenia:**  xxxxx  **Prowadzący ćwiczenie:**  xxxxx |

1. **Treść zadania laboratoryjnego**

Obraz zawierający tekst

Opis wygenerowany automatycznie

1. **Opis rozwiązania, komentarze, wnioski**

Plik producenta („p.c”):

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <signal.h>

#include <errno.h>

#define STREAM\_SIZE 120

struct mymsgbuf {

  pid\_t mtype;

  char value[STREAM\_SIZE];

} queue;

void send\_message (int qid, pid\_t target, char \*content) {

  queue.mtype = target;

  strcpy(queue.value, content);

  if (msgsnd(qid, &queue, sizeof(struct mymsgbuf) - sizeof(pid\_t), 0) == -1) {

    fprintf(stderr, "Failed to send message\n");

    exit(EXIT\_FAILURE);

  }

};

int main (int argc, char \*argv[]) {

  pid\_t k1, k2, k3;

  if ((k2 = fork()) == 0) {

    execlp("./k2.exe", "k2", (char \*)NULL);

  } else if ((k3 = fork()) == 0) {

    execlp("./k3.exe", "k3", (char \*)NULL);

  } else if ((k1 = fork()) == 0) {

    execlp("./k1.exe", "k1", (char \*)NULL);

  } else {

    if (argc != 2) {

      fprintf(stderr, "Invalid arguments length\n");

      exit(EXIT\_FAILURE);

    }

    FILE \*f = fopen(argv[1], "r");

    if (f == NULL) {

      fprintf(stderr, "Failed to open file\n");

      exit(EXIT\_FAILURE);

    }

    int qid;

    key\_t msgkey = ftok(".", 'm');

    if ((qid = msgget(msgkey, IPC\_CREAT | 0660 )) == -1) {

      fprintf(stderr, "Failed to open message queue\n");

      exit(EXIT\_FAILURE);

    }

    char \*line = NULL;

    size\_t len = STREAM\_SIZE;

    ssize\_t read;

    printf("Starting reading file...\n");

    while ((read = getline(&line, &len, f)) != -1) {

      send\_message(qid, k1, line);

      send\_message(qid, k3, line);

      send\_message(qid, k2, line);

      //sleep(1);

    };

    fclose(f);

    sleep(5); // wait for child processes to execute

    kill(k1, 9);

    kill(k2, 9);

    kill(k3, 9);

    if (msgctl(qid, IPC\_RMID, 0) == -1) {

      fprintf(stderr, "Failed to cleanup queue\n");

      exit(EXIT\_FAILURE);

    }

  }

  return EXIT\_SUCCESS;

};

Plik konsumenta1 („k1.c”):

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <errno.h>

#define STREAM\_SIZE 120

struct mymsgbuf {

  pid\_t mtype;

  char value[STREAM\_SIZE];

} queue;

struct mymsgbuf readbuffer;

int main (void) {

  int qid;

  key\_t msgkey = ftok(".", 'm');

  if ((qid = msgget(msgkey, IPC\_CREAT | 0660)) == -1) {

     fprintf(stderr, "Failed to open message queue\n");

     exit(EXIT\_FAILURE);

  }

  readbuffer.mtype = getpid();

  while (msgrcv(qid, &readbuffer, sizeof(struct mymsgbuf) - sizeof(pid\_t), readbuffer.mtype, IPC\_NOWAIT) != -1) {

    printf("[K1] Odczytano: %s", readbuffer.value);

    sleep(1);

  };

  return EXIT\_SUCCESS;

};

Plik konsumenta2 („k2.c”):

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <errno.h>

#define STREAM\_SIZE 120

struct mymsgbuf {

  pid\_t mtype;

  char value[STREAM\_SIZE];

} queue;

struct mymsgbuf readbuffer;

int main (void) {

  int qid;

  key\_t msgkey = ftok(".", 'm');

  if ((qid = msgget(msgkey, IPC\_CREAT | 0660)) == -1) {

     fprintf(stderr, "Failed to open message queue\n");

     exit(EXIT\_FAILURE);

  }

  readbuffer.mtype = getpid();

  while (msgrcv(qid, &readbuffer, sizeof(struct mymsgbuf) - sizeof(pid\_t), readbuffer.mtype, IPC\_NOWAIT) != -1) {

    printf("[K2] Odczytano: %s", readbuffer.value);

    sleep(1);

  };

  return EXIT\_SUCCESS;

};

Plik konsumenta3 („k3.c”):

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <errno.h>

#define STREAM\_SIZE 120

struct mymsgbuf {

  pid\_t mtype;

  char value[STREAM\_SIZE];

} queue;

struct mymsgbuf readbuffer;

int main (void) {

  int qid;

  key\_t msgkey = ftok(".", 'm');

  if ((qid = msgget(msgkey, IPC\_CREAT | 0660)) == -1) {

     fprintf(stderr, "Failed to open message queue\n");

     exit(EXIT\_FAILURE);

  }

  readbuffer.mtype = getpid();

  while (msgrcv(qid, &readbuffer, sizeof(struct mymsgbuf) - sizeof(pid\_t), readbuffer.mtype, IPC\_NOWAIT) != -1) {

    printf("[K3] Odczytano: %s", readbuffer.value);

    sleep(1);

  };

  return EXIT\_SUCCESS;

};

Każdy konsument po odczytaniu danej odczekuje 1 sekundę. Umożliwia to odpowiednią synchronizację procesów przez producenta

Po odczytaniu wszystkich danych z pliku tekstowego proces producenta usuwa wszystkie kolejki komunikatów

Działanie kolejek komunikatów:

1. Tworzenie kolejki (msgget)
2. Ustawianie typu wiadomości (wartość dodatnia)
3. Ustawianie wartości komunikatu (zmienna)
4. Wysyłanie komunikatu do kolejki (msgsnd)
5. Odczytanie z kolejki (msgrcv)
6. Usuwanie kolejki po zakończeniu pobierania (msgctl)
7. **Wyniki uruchomienia programu**

Obraz zawierający tekst, tabliczka

Opis wygenerowany automatycznie