Betriebssysteme WS21   
Praktikum Aufgabe 3   
Prozesse und Kommunikation über Message Queues

Bearbeitet von:

|  |  |
| --- | --- |
| Name: | Maximilian Mayer |
| Matrikel-Nr.: | 91043 |
| Studiengang: | Technische Informatik Berufsbegleitend |
| Semester: | WS21 |

**Inhaltsverzeichnis**

[1 Vorgehensweise zur Lösung von A3 1](#_Toc92115725)

[1.1 Aufbau der int main (void) 1](#_Toc92115726)

[1.2 Aufbau der Funktion int fillContacts (int numberOfContacts) 2](#_Toc92115727)

[1.3 Aufbau der Funktion void clientCode (int clientNr) 2](#_Toc92115728)

[1.4 Aufbau der Funktion void serverCode (void) 2](#_Toc92115729)

[1.5 Aufbau der Funktion static void sigChildHandler (int signr) 3](#_Toc92115730)

[2 Hilfe zum debuggen 3](#_Toc92115731)

[3 Dokumentation der Programmausführung 4](#_Toc92115732)

[3.1 Nachtrag 6](#_Toc92115733)

[4 Sequenzdiagramm 8](#_Toc92115734)

[5 Antworten zu den Fragen im Aufgabendokument 9](#_Toc92115735)

# Vorgehensweise zur Lösung von A3

## Aufbau der int main (void)

Als Basis habe ich das zu Verfügung gestellte Rumpfprogramm verwendet.  
In der main des Rumpfprogramms sind bereits einige ToDos beschrieben.  
Nachdem die Variablen initalisiert wurden und die „Kontakt-Datenbank“ aus „zufälligen“ Daten erzeugt wurde müssen die Message Queues eingerichtet werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Es wird eine Message Queue askContact erzeugt und Anzahl n resultContacts Message Queues die über die Vorschleife in Abhängigkeit von der Konstanten MAXCLIENTS erzeugt werden.Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Als nächstes wird die Semaphore die für das printf verwendet werden soll eingerichtet.

Danach muss der Handler für SIGCHLD eingerichtet werden.

Dies wird über die Funktion initHandler realisiert.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Als nächstes müssen die Clients erzeugt werden.  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung  
Mittels for-Schleife werden Abhängig von der Konstanten MAXCLIENTS, Prozesse erzeugt welche die Funktion clientCode ausführen. Außerdem wird die Variable nClients hochgezählt. Dort wird die Anzahl der Clients gespeichert.

Zuletzt muss noch die Funktion serverCode aufgerufen werden.  


## Aufbau der Funktion int fillContacts (int numberOfContacts)

Diese Funktion wurde bereits zur Verfügung gestellt.  
Abhängig von der Variablen numberOfContacts welche mit der Funktion übergeben werden muss. Wird eine „Datenbank“ aus Vornamen, Nachnamen und Email-Adressen erzeugt. Diese Daten werden „züfällig“ erzeugt. Als Rückgabewert erhält man die Anzahl der erzeugten „Datensätze“.

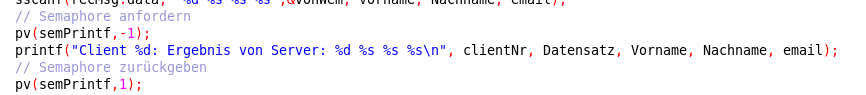
## Aufbau der Funktion void clientCode (int clientNr)

Nach jedem erfolgreichen Fork welcher ausgeführt wird, wird die Funktion clientCode ausgeführt. Die Funktion bekommt die Variable nClients übergeben welche bei jedem erzeugen eines Prozesses hochgezählt wird.

Die Funktion wurde zum Großteil ebenfalls bereitgestellt. Abhängig von der Konstanten Anzahl schickt jeder Client eine Message an den Server. Diese Message setzt sich zusammen aus der Variablen vonWem (in dieser Variablen wird die ClientNr gespeichert) und der Variablen Datensatz (in dieser Variablen wird eine zufällige Zahl gespeichert).   


Dann wartet der Prozess bis dieser eine Message (resultContacts) vom Server erhält.  


Nachdem die Nachricht empfangen wurde, wird diese entsprechend zerlegt.  
  
In der Nachricht ist enthalten von wem die Nachricht ist, Vorname, Nachname sowie die Email.

Nachdem die Semaphore angefordert wurde, wird die ClientNr, der Datsensatz, sowie der entsprechende Vorname, Nachname und die Email-Adresse auf der Konsole ausgegeben.  


Sobald die Anzahl der entsprechenden Messages empfangen, verarbeitet und ausgegeben wurde, beendet sich der Client.

## Aufbau der Funktion void serverCode (void)

Auch bei der Funktion servcerCode wurde schon einiges zur Verfügung gestellt.  
Der Eltern-Prozess befindet sich in einer while(1) Schleife in der er auf Nachrichten der Clients wartet.  


Sobald eine Nachricht empfangen wurde wird diese zerlegt und auf der Konsole ausgegeben, bevor die Ausgabe durchgeführt wird, wird noch die Semaphore angefordert und nach der Ausgabe zurückgegeben.  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Um dem Client den entsprechenden Datensatz senden zu können muss die Nachricht noch zusammengesetzt werden. Anhand der Variablen Datensatz kann der entsprechende Vorname, Nachname und die Email Adresse in die Variablen kopiert werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Mittels msgsnd wird dann dem entsprechenden Client die Nachricht resultContacts gesendet.  


Nachdem dies geschehen ist, wartet der Server auf die nächste Nachricht.

## Aufbau der Funktion static void sigChildHandler (int signr)

Sobald eine Client beendet wird, wird diese Funktion aufgerufen.  
Es wird auf der Konsole ausgegeben das der sigChildHandler ausgeführt wird.  
Außerdem wird die Variable nClients um 1 reduzuert.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung  
Sobald die Variable nClients==0 ist, werden die Message Queues gelöscht. Außerdem wird die Semaphore zerstört und der Elternprozess beendet.

# Hilfe zum debuggen

Um bei dem Erstellen des Programms dem Ablauf besser folgen zu können, habe ich einige printf im Code eingefügt.

Dazu habe ich mir das define DEBUG angelegt.  
Ist das define aktiv, wird über ifdef DEBUG die Ausgabe „aktiviert“.   


# Dokumentation der Programmausführung

Um das Programm besser dokumentieren zu können habe ich die define DEBUG aktiviert.   
Außerdem sind die Konstanten wie folgt gesetzt.  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung  
Es werden also 5 Clients erzeugt die jeweils 3 Messages senden.

Im folgenden Screenshot ist der Ablauf des Programmes zu sehen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* Die erste Ausgabe gibt Informationen über den Erstellers des Codes
* Elternprozess mit der PID
* Ausgabe über die Konstanten MAXCLIENTS und ANZAHL der Nachrichten
* Zeigt die angelegten Kontakte (Anzahl der Kontakte kann über Konstante CONTACT\_N eingestellt werden)
* Außerdem wir die Kennung der Semaphore ausgegeben

Im folgenden Bildschirmausschnitt wird das starten der einzelnen Clients ausgegeben. Client n: nach Fork. Außerdem der Beginn des Clients.   
Der Client sendet dann an den Server eine Nachricht, Server: Frage von Client n nach Datensatz n.  
Der Server Antwortet, wobei der entsprechende Client dann den Datensatz ausgibt.

Die unterschiedlichen Nachrichten habe ich farblich markiert um zu sehen welche Nachrichten von den einzelnen Clients ausgelöst werden.  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Nachdem alle Anfragen und Antworten der Clients verarbeitet wurden beenden sich diese.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Sobald ein Client beendet wurde, wird der Code im SigChildHandler ausgeführt.

Wie im Bildschirmausschnitt zu sehen ist, wird nach dem beenden der Clients nochmals die Nachricht „Server: Frage von Client 5 nach Datensatz 0“ ausgegeben. Dies ist laut Aufgabenstellung nicht gewünscht. Leider konnte ich diesen „Fehler“ nicht beseitigen.  
Meine Vermutung ist das wenn ein Client beendet wird ja der sigChildHandler abgearbeitet wird. Sollte nClients nicht == 0 sein, dann kommt in der Funktion das return; zum tragen. Das bedeutet der Code läuft im Servercode weiter. Da der Code auf eine Nachricht wartet (msgrcv) läuft der Code weiter. Es wird also wieder das printf ausgeführt. Dann wartet das Programm wieder auf die nächste Nachricht, bzw. darauf das ein Client sich beendet.

## Nachtrag

Im Servercode konnte ich durch das einfügen einer Fehler- bzw. Unterbrechungsbehandlung verhindern das Nachrichten welche mit einem Fehlercode abgefangen werden.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Außerdem konnte ich den Fehler abfangen falls der Signalhandler ein Signal „verpasst“.  
Dazu habe ich den sigChildHandler wie folgt angepasst.  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In meinen Tests konnte ich nun keine „falsche“ Server-Ausgabe (Server: Frage von Client n nach Datensatz n) mehr feststellen.  
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Sollte der Signalhandler ein Signal „verpassen“ wird der Zombie mit waitpid gelöscht.  
‚Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

# Sequenzdiagramm

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

# Antworten zu den Fragen im Aufgabendokument

**- Welchen Speicherbedarf hat ein Objekt vom Typ contactsStruct?**   
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung  
Ein int benötigt 16Bit Speicher.  
Ein char benötig 8Bit  
In diesem Fall nehmen wir an das CONTAC\_N = 2 und NAMELENGHT = 15 ist.  
Somit hat char Nachname einen Speicherbedarf von 8Bit \* 15 pro CONTACT\_N. In diesem Fall also 8Bit \* 15 \* 3 = 360Biit

Das selbe für char Vorname. 8Bit \* 15 \* 3 = 360Bit

Char email hat eine NAMELENGTH von 3 \* 15 = 45. Das ergibt 8Bit \* 45 \* 3 = 1080Bit

Im Gesamten:  
int couter = 16Bit  
char Nachname = 360Bit  
char Vorname = 360Bit  
char email = 1080Bit

Gesamt = 1816Bit  
  
**- Wie lange dauert die komplette Ausführung des Programms vom Start bis zum Ende aller Prozesse?**   
Bei MAXCLIENTS = 5 und Anzahl Messages = 3 und einer SLEEPTIME von 20000 Microsekunden benötigt das Programm 0,069s

