Operativsystemer og Multiprogrammering
G-opgave 4

Ronni Elken Lindsgaard - 0911831791
Hans-Kristian Bjerregaard - 0612862087
Alexander Winther Uldall - 2908872013

16. marts, 2010

1 othread lageradministration

1.1 Strategi

Vi har valgt at holde 3 typer af lister for let at kunne hive de informationer ud som skal bruges. En global liste holder styr på alle tråde der pt er allokerede som deles mellem processer. En liste for hver tråd som holder styr på de af `othread_malloc` allokerede områder som tråden har adgang til. Den sidste liste ligger i en datastruktur før selve det allokerede område med metadata. Denne liste indholder en liste over samtlige tråde der henviser til dette område.

1.2 `othread_malloc`

Først tjekkes om det område defineret i `memid` ved at løbe igennem den globale liste over allokerede områder. Findes det allerede returneres pointeren til det i forvejen allokerede sted i hukommelsen. Hvis det ikke findes, allokeres et nyt område. De tre lister nævnt ovenfor opdateres med undtagelsen hvis `memid` er 0 da den globale liste ikke opdateres for at sikre private område. Til sidst initialiseres den allokerede hukommelse til `NULL` da vi derfor kan være sikre på data's værdi.

1.3 `othread_free`

Listeelementerne i tråden og datastrukturens liste bliver fjernet med det samme da disse ikke længere er skal bruges. Herefter tjekkes om der stadig er tråde der peger på den delte hukommelse, hvis ikke bliver området fjernet fra den globale liste og hele området deallokeres.

`othread_exit` er blevet ændret således at for alle adresserum der peges på i listen af allokerede adresserum bliver der kaldt `othread_free` - så længe at `othread_exit` kaldes ved afslutningen af hver tråd vil altså alle allokerede områder også blive deallokerede.

1.4 Kritiske regioner

Ved tvungen trådskift kan vi risikere at skifte midt i udførslen af koden. Der er flere lister der skal opdateres, hvis et trådskift forekommer midt i opdateringen af disse lister vil eventuelle tråde der bliver eksekverede arbejde på forkert hukommelse. Det er derfor nødvendigt at både `othread_malloc` og `othread_free` udføres atomart. De kritiske regioner består derfor af hele `othread_malloc` og `othread_free`.

1.5 Kontrol af rigtighed

Allokeringer med samme memid skal pege samme sted hen både i samme tråd og mellem tråde. Dette er gjort ved at dele en tæller-variabel mellem trådene Allokeringer med memid=0 skal være privat allokerede kun for den specifikke tråd. Dette er gjort ved arbejde med variablen på samme måde og se at ændringer kun gælder for den enkelte tråd. For `othread_free` skal det være muligt at tilgå hukommelsesområdet så længe der er mindst een tråd der peger på det allokerede område. Dette skal derfor aldrig være tilfældet for `memid` = 0.

Delte allokeringer skal være tilgængelige for forskellige typer af tråde imellem, dette er kun gældende for allokeringer hvor `memid` $\neq 0$.

Til sidst skal der tjekkes om hukommelsen rent faktisk bliver frigjort når trådene exiter. Dette er gjort vha. programmet valgrind og resultatet er at der ikke forefindes allokeret hukommelse efter programmets afslutning. Testprogrammet hedder `malloc.c`.

2 Segmenterede sidetabeller

Her følger den generiske algoritme for at konvertere en logisk adresse til en fysisk adresse på en Intel Pentium processor. De første 4 punkter dækker segmenteringen mens de sidste 3 dækker over sideopslag. Navne i algoritmen refererer figurerne på side 346 (figurerne er unummererede) i SGG - 8. udgave.

- Brug g i selector til at bestemme om GDT eller LDT skal benyttes.
- Benyt s til at slå base og limit op i GDT/LDT.
- Tjek at offset er mindre end limit, en fejl opstår hvis dette ikke er tilfældet.
- Beregn 32-bit lineær adresse ved at summere offset og base.
- Benyt de første 10 bit (p_1) i den lineære adresse til at så op i sidekataloget for at finde den anvendte sidetabel.
- Benyt de efterfølgende 10 bit (p_p) til at slå op i den fundne sidetabel.
- Den fysiske adresse findes så ved at tage summen af resultatet i sidetabelen og den lineære adresses offset (d).

2.1

selector: 0x270, offset 0x10

Selectoren i binær er 1001111000 hvilket giver $p = 00$, $g = 0$ og $s = 1001111$. Da g er nul benyttes GDT. Decimalværdien af s er 78 og på position 78 i GDT

finder vi basen 0x803000 og limit 0x100. Da limit er større end offsetet i den logiske adresse kan vi fortsætte. Så beregnes den lineære adresse som summen af det logiske offset og basen hvilket giver 0x803010.

Den binære værdi af den lineære adresse er 100000000011000000010000 hvilket giver $d = 000000010000$, $p_2 = 0000000011$ og $p_1 = 10$. Decimalværdien af p_1 er 2 hvilket benyttes til at slå op i sidekataloget. Decimalværdien af p_2 er 3 hvilet bruges til at slå op i den sidetabel fundet i sidekataloget. Den fysiske adresse er så sidens adresse plus d hvilket giver: 0x07 + 000000010000 = 10111.

2.2

selector: 0x278, offset 0xFA

Selectoren i binær er 1001111000 hvilket giver $p = 00$, $g = 0$ og $s = 1001111$. Da g er nul benyttes GDT. Decimalværdien af s er 79 og på position 79 i GDT finder vi basen 0xC01000 og limit 0x80. Da limit er mindre end offsetet i den logiske adresse kan vi ikke fortsætte!

2.3

selector: 0x10C, offset 0xFFF

Selectoren i binær er 100001100 hvilket giver $p = 00$, $g = 1$ og $s = 100001$. Da g er et benyttes LDT. Decimalværdien af s er 33 og på position 33 i LDT finder vi basen 0xC00000 og limit 0x1000. Da limit er større end offsetet i den logiske adresse kan vi fortsætte. Så beregnes den lineære adresse som summen af det logiske offset og basen hvilket giver 0xC00FFF.

Den binære værdi af den lineære adresse er 110000000000111111111111 hvilket giver $d = 111111111111$, $p_2 = 0000000000$ og $p_1 = 11$. Decimalværdien af p_1 er 3 hvilket benyttes til at slå op i sidekataloget. Decimalværdien af p_2 er 0 hvilet bruges til at slå op i den sidetabel fundet i sidekataloget. Den fysiske adresse er så sidens adresse plus d hvilket giver: 0x43 + 000000010000 = 1010011.