Mittuniversitetet
MID SWEDEN UNIVERSITY

Laboration 3 - Minnet

Operativsystem introduktionskurs

Jimmy Åhlander*

15 december 2021

Innehåll

1 Introduktion

Primärminnet, *arbetsminnet*, eller ibland lite slarvigt *"RAM-minnet"* eller bara *minnet* är det minne som processer använder för att i stunden lagra sin kod och data. Primärminnet är flyktigt men förhållandevis snabbt om än med en lägre kapacitet jämfört med sekundärminnet. Moderna minnesarkitekturer är

*jimmy.ahlander@miun.se. Avdelningen för informationssystem och -teknologi (IST)

invecklade och använder olika former av caches och sekundärlagring för att öka primärminnets effektivitet och kapacitet.

I den här laborationen kommer du med hjälp av `C++` att utforska de begränsningar som föreligger i ett Windowssystem för att besvara frågan: Hur mycket minne kan en process maximalt använda?

2 Mål

Efter genomförd laboration ska du ha en bättre förståelse för hur ett modernt operativsystem hanterar primärminnet.

Följande lärandemål är kopplat till laborationen. Du ska kunna:

- redogöra för den principiella funktionen hos de viktigaste logiska delarna som t.ex. minnes- och processhantering i ett operativsystem och förklara deras inbördes relationer.
- identifiera och redogöra för betydelsen av några viktiga parametrar för prestanda i ett operativsystem.

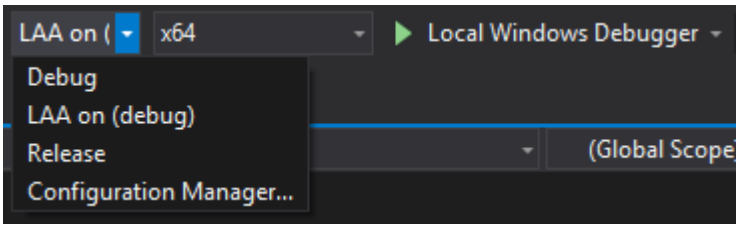
3 Teori

Hur mycket minne en process kan använda beror på flera faktorer, bland annat den tillgängliga adressrymden, till exempel 32 bitar eller 64 bitar, som i sin tur begränsas av processorarkitekturen, exempelvis `x86` eller `x86-64`. En annan faktor är om *virtuellt minne* nyttjas. Utan virtuellt minne är processen begränsad till det installerade fysiska minnet. Med virtuellt minne är datalagringen istället abstraherad och processen arbetar i en logisk minnesrymd som är separerad från sin fysiska motsvarighet genom tekniker som *paging* och *segmentering*. Detta tillåter processen att i nödfall använda diskutrymme som ett substitut för primärminnet. Detta vet förstås inte processen om utan hanteras helt av operativsystemet. En tredje faktor är vilken konfiguration som används för den exekverbara filen. Det finns olika sätt att kommunicera till operativsystemet hur mycket minne som ska reserveras för olika ändamål, till exempel med en flagga som *Large Address Aware* (LAA). Det finns också olika sätt att själv begränsa den mängd minne som processen kan nyttja.

4 Genomförande

Din uppgift är att skapa ett program i `C++` som successivt allokerar dynamiskt minne i chunks tills att processen når en brytningspunkt där det inte längre går att allokera mer minne. Målet är att hitta denna brytpunkt för olika konfigurationer och målplattformar.

Laborationen är avsedd att genomföras i Windowsmiljö med hjälp av Visual Studio *(inte Visual Studio Code)*. Visual Studio inkluderar verktyg för att övervaka process- och minnesnyttjande samt tillåter enkelt skifte mellan olika konfigurationer och målplattformar, se Figur 1, vilket är vitalt för denna uppgift. All nödvändig mjukvara finns i Windowslabben på campus.



Figur 1: Du kan skapa nya konfigurationer genom VS Configuration Manager.

Generellt kommer de flesta projekt med konfigurationerna `Debug` och `Release`. Du bör även ha två färdiga målplattformar för `x86` och `x64`. För enkelhets skull bör du skapa ett par nya konfigurationer för LAA på/av. Du hittar inställningarna för LAA i projektegenskaperna.

Program

Programmet skapas i `C++`. Viktiga nyckelord är `new[]`, `try`, `catch`, `sizeof()` och fixed width integer types som `int64_t` [1].

I korthet ska ditt program:

1. Visa en meny för val av chunkstorlek.
2. Köra en oändlig loop, där det ...
   (a) försöker allokera en chunk minne genom en dynamisk array med hjälp av `new[]`,
   (b) skriver ut hur mycket minne som hittills allokerats i arrayerna, och
   (c) fångar eventuell `bad_alloc` med `catch`.

Programkoden blir därför inte särskilt lång. En cpp-fil räcker.

Test

Ditt program ska du använda i ett antal test vars resultat du dokumenterar i en labbrapport.

Du ska testa följande chunkstorlekar:

- 1 GiB.[1]
- 1 KiB.

mot följande:

- 32-bitars målplattform.
- 64-bitars målplattform.

i följande konfigurationer:

- med large addresses.[2]
- utan large addresses.

Totalt får du därför $2^3 = 8$ testfall att köra och evaluera. Ställ in VS på den konfiguration och målplattform du vill testa. Sätt upp din övervakning och låt därefter datorn arbeta ostört medan programmet körs då dina andra aktiviteter kan påverka testresultatet[3].

För varje testfall dokumenterar du följande:

1. Vilket fel du får när processen inte kan allokera mer minne. Vad händer?
2. Hur mycket minne processen och systemet använt.

   Mät med aktivitetshanteraren (under prestanda – minne), den inbyggda övervakningen i VS och dina egna utskrifter från programmet.
3. Övrig relevant prestandadata med valfria verktyg, t.ex. disk- och processornyttjande samt hur lång tid körningen tar. Observera även minneskompositionen genom aktivitetshanteraren.

OBSERVERA: När du arbetar med stora mängder minne kan systemet bli instabilt och i värsta fall krascha. Spara ditt arbete regelbundet.

[1] Kom ihåg att i Microsofts värld så presenteras 1 GiB som 1 GB, 1 KiB som 1 KB, osv.
[2] Large Address Aware (LAA)-flaggan på.
[3] Att försöka starta YouTube på en dator som allokerat 2 gånger sitt fysiska minne är som att hoppa i ett badkar som är fyllt hela vägen till kanten.

Analys

Efter körningarna analyserar du resultaten. En analys besvarar varför du får det resultat du får. Följande frågor kan användas som inspiration:

- Hur och varför skiljer sig värdena från de olika verktygen?
- Vilka teoretiska och praktiska gränser finns för minnesallokering? Hänvisa till externa källor.
- Hur förändrades minneskompositionen under körning och varför?
- Vad har virtuellt minne för effekt?

Testresultaten och din analys sammanställer du i en labbrapport. Presentera resultaten med hjälp av **text**, en **tabell** och en eller ett par **välstrukturerade diagram**. Rapporten bör vara 1-2 sidor, exklusive figurer och källförteckning.

Kom ihåg att även dokumentera förutsättningarna för dina tester: Relevant systemdata som OS, hur mycket primärminne som finns installerat, hur mycket primärminne som var ledigt vid startet av testerna och liknande.

Tips

`new[]` skapar en array med ett givet antal element av den valda datatypen. Välj din datatyp omsorgsfullt och beräkna antalet element tillsammans med `sizeof()` för att försäkra dig att chunkstorleken blir rätt.

Strunta i pekare och `delete[]`. Detta kommer förstås leda till minnesläckor men det spelar ingen roll i denna uppgift – du vill bara allokera minne tills att det spricker.

Pausa exekveringen på valfritt sätt när du fångar `bad_alloc` så kan du mäta maxvärdena i lugn och ro. När processen terminerar frigörs minnet automatiskt.

Använd modulus för att undvika onödigt många utskrifter när du allokerar små chunks. Det räcker om du har en granularitet på 1 MiB i dina utskrifter.

5 Examination

Redovisa laborationen under ett av laborationstillfällena som ges under kursens gång genom att visa din programkod och färdiga labbrapport. När du fått klartecken från labbhandledaren att redovisningen är godkänd kan du lämna in i inlämningslådan på lärplattformen.

Din inlämning ska bestå av en cpp-fil för programkoden och en PDF-fil för rapporten. Inga ZIP-arkiv eller andra dokumentformat accepteras.

5.1 Bedömning och återkoppling

Uppgiften bedöms med betygen *Godkänd (G)*, *Komplettering (Fx)*, och *Underkänd (U)*. Bedömningen baseras i huvudsak på huruvida du under den muntliga delen av examinationen kan förklara din programkod, om dina testresultat är korrekta och om din analys är tillräcklig.

För godkänt betyg måste din rapport inkludera namn, program, datum, efterfrågad tabell och diagram samt åtminstone någon extern källa i analysen med källförteckning.

Återkoppling erhåller du i första hand muntligt från labbhandledaren under och efter redovisningen. Normalt lämnas ingen skriftlig återkoppling vid godkänt resultat för denna uppgift.

Referenser

[1] cppreference contributors, “Fixed width integer types.” [Online]. Available: https://en.cppreference.com/w/cpp/types/integer