Homework#2 (h2) Latest Version3: 8 pm, Oct 3 CSC148 Prof Mitchell

Required HandIns section added in Version3 Due date On/Before 11:59pm Thursday October 8

#1 Introduction

A first example hps model was covered in class. It is a special case of hps service because each cj requests exactly one server. In many real systems (equipment rentals, thread allocation to computer system tasks, etc.) , cj requests a random (small) number of servers rather than just one server.

In h2, the following significant extensions of the hps covered in class are implemented:

1. Each arriving cj requests a (small) random number of free servers from the hps degree total
2. Two different cj service configurations are compared:

* In the first configuration, cj service duration is not affected by current hps “load”, where

load is defined as the number of busy servers – the non-congestion model

* In the second configuration, cj service duration is dependent on current hps load - the congestion model

Service duration is slower or faster depending on whether many vs. very few servers are busy; the second configuration is more realistic; real hps systems use internal resources for implementing services, and such resources are always limited in physical systems *(Example: in a multi-user & multi-core computer server, slow response means many users are competing for memory and execution time slices)*

#2 cj requests randomly many servers

If cj is allocated x free servers using LEAVE, cj must, before termination, give back the x servers with an ENTER block.

Some thought should convince you that using a savevalue to store x during cj’s lifetime is not sufficiently general.

The reason is that each cj will need a savevalue to store its random x value. For large hps degree, and unpredictable queueLength, the number of savevalue entities needing an initial statement is unpredictable

=> many savevalues are never used OR the number of savevalue INITIAL statements is exceeded in long model runs.

Using a tr parameter solves the SAVEVALUE dilemma:

Each cj uses a “**tr parameter”** for storing its x value. Each tr parameter is a “private local variable” to that tr, and exists +only for the duration of that tr.

The only way in gpss that a tr can store a value in a tr parameter is with: assign parmName,value

The way that the value currently stored in a tr parameter is retrieved is with the expression p$parmName.

In summary, the life-cycle of a tr parameter (suppose it is named “myparm”) is:

1. An EQU statement defines a positive int value for myparm
2. Use an ASSIGN block to store a value in myparm (one or more ASSIGN blocks per tr, as needed in a model)
3. The current value stored in myparm is retrieved by its tr using the expression p$myparm

Part of tr occurrence’s local address space:

parameter#1

parameter#2, etc.

A tr instance “R” has no parameters when it starts, and each R ASSIGN block stores a value in an R parameter. All of R’s parameters are destroyed when R terminates.

**DEMO**

; **demoTrParameters.gps**

; Demo of defining and using a tr parameter

clear

orderSize EQU 1 ; order tr parameter for cj’s random order size

initial x$ordersTotal,0 ; Initialize sum of order sizes to 0

order generate 5,,,,1 ; An order tr every 5 t.u., of higher priority than snoop tr

assign orderSize,(RN2@10+1) ; orderSize is a random variable (integer) in [1,10]

advance 3

savevalue ordersTotal+,p$ordersize ; Increment order sizes total

terminate 1 ; This order tr finishes

snoop generate 10 ; One snoop every 10 t.u.

; In snoop, orderSize refers to tr snoop's, NOT an order tr’s, parameter

; Also, tr parameters are dynamically created at first reference, so at THIS POINT, before the

; ASSIGN, the reference p$orderSize causes a “fatal’ (gpss words) run error, but the model run CAN BE Continued

assign orderSize,-83 ; This snoop tr's orderSize parameter gets its own private value

terminate 1 ; This snoop tr finishes

*Note - A coding/readability requirement is: use a user-defined symbol for each tr parameter. Do not code legal, but meaningless names such as p17, or the integer 17 when referencing the tr parameter.* *Doing so is poor source code documentation.*

#3 Modeling hps congestion

During model execution, the gpss built-in function call r$hpsName returns the number of busy servers, where hpsName is the label on the STORAGE statement associated with this hps *(that is, the hps’s name in the source code)*

Use the following expression for additional service duration for cj when the hps servers compete for resources: (.45)**(r$hpsName)2/degree abbreviate this expression as “sc”**. *The value of sc is interpreted as sc t.u.s*

*Note that the following inequalities hold: 0 <=* ***sc*** *<= (.45) ( (number of busy servers)2 )/degree < = (.45) degree*.

Coding note **-** unlike many languages, neither juxtaposition (such as 3ABC) nor \* (such as 3\*ABC) are the multiplication operator. The default (that you can change using the gpssW Settings menu) is that # is the multiplication operator.

SNA note- The SNAs (system numerical attributes, i.e. built-in functions in most language terminology) for a STORAGE statement include: s$hpsName returns the number of free servers and r$hpsName returns the number of busy servers

There are 2 (operational) Configurations / Use Cases for cj service)

A ) Intra-server congestion is not represented ; service duration is modeled as in module P2 Slides 7-9.

B ) Intra-server congestion is modeled by adding term sc to the Exponential() service duration call used in A )

#4 Model inputs

In each model run, the random number of allocated servers to each cj is the expression RN2@3+1

All runs use exponential {ia} and service duration distributions (Yellow part of posted h1 EXTENDED solution code)

Each model run uses: fixed {ia} mean = 1/ = 1.0 t.u., and service duration mean = 1/t.u., and Start 10000

EVERY run uses DIFFERENT RN1, RN2 4-digit) seed pairs = > avoids reliance on one RNk for pseudo random numbers.

#5 HandIns \* See the un-numbered Note at the end of the document

First HandIn – Fill in Table 1 and Table 2 as described below

Make runs, starting with degree = 8, and double the degree on successive runs. Continue until a run result has L <= 4.0 In the row for each run’s results fill in the third column (as needed) and columns 5 through last column.

Table 1 – Runs without server congestion

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ****  **fixed** | ****  **0.25 fixed** | **hps degree** | **gpss result ** | **avg(Wcj)** | **L = avg(n(t))** | **MAX**  **L in S** | **RN1 seed** | **RN2 seed** |
| **Fixed** | **Fixed** | **8** |  |  |  |  |  |  |
| **Fixe** | **Fixed** | **16** |  |  |  |  |  |  |
| **Fixed** | **Fixed** | **32** |  |  |  |  |  |  |

Make runs starting, with degree = 8, and quadruple the degree on successive runs. Continue until a run occurs with L <= 5.0 The number of runs needed is unpredictable. Fill in Table 2 as done for Table 1.

Note!!! A degree 8 run in Table 2 can take well over 10 seconds, depending on your machine’s speed.

Table 2 – Runs with server congestion

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ****  **fixed** | ****  **0.25 fixed** | **hps degree** | **gpss result ** | **avg(Wcj)** | **L =**  **avg(n(t))** | **MAX**  **L in S** | **RN1 seed** | **RN2 seed** |
| **Fixed** | **Fixed** | **8** |  |  |  |  |  |  |
| **Fixed** | **Fixed** | **32** |  |  |  |  |  |  |
| **Fixed** | **Fixed** | **128** |  |  |  |  |  |  |
| **Fixed** | **Fixed** | **512** |  |  |  |  |  |  |

Second HandIn - For the congestion Configuration hps degree 32 only

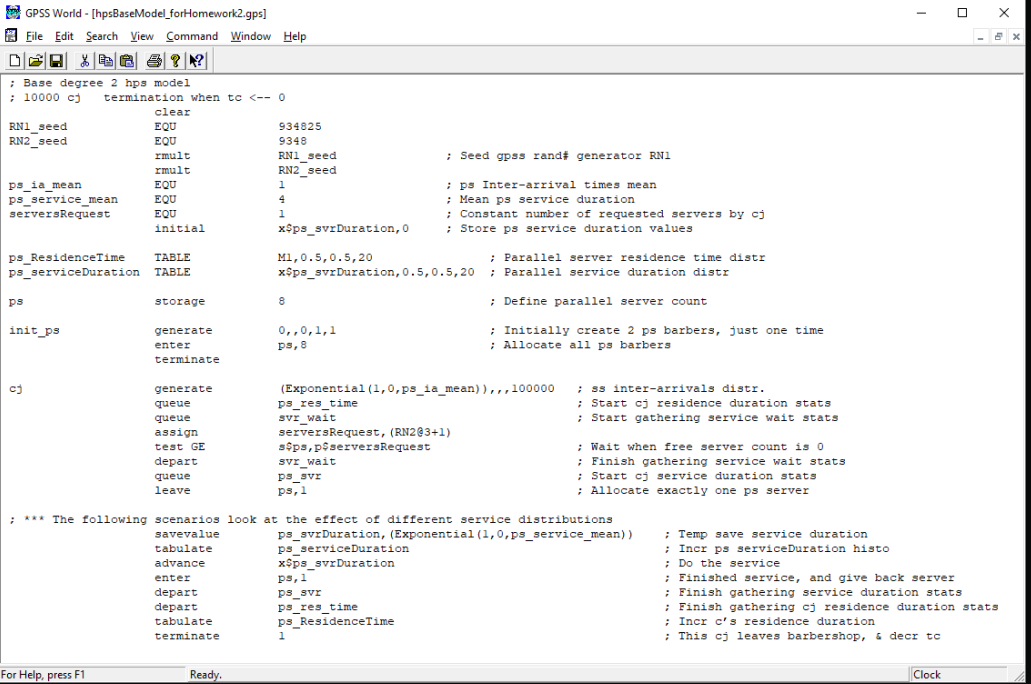
a) the Report page QUEUE and STORAGE sections only – just the 6 or 7 or report lines; DO NOT submit that entire Report file that might be several pages

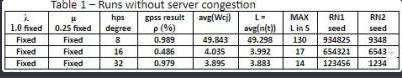
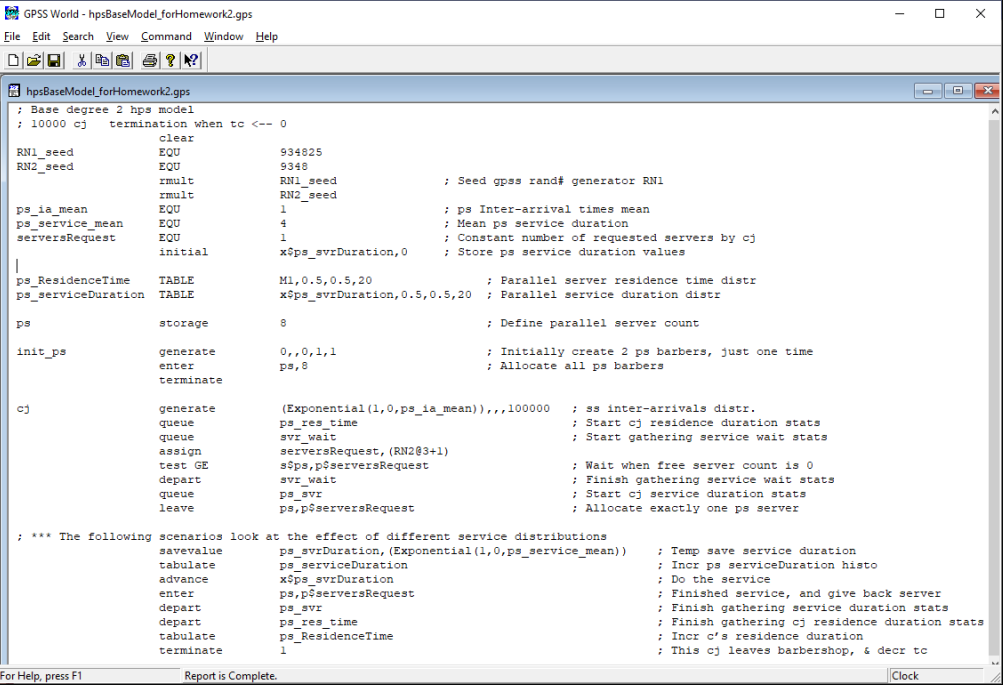
b) The histogram of L values captured when each cj enters the model

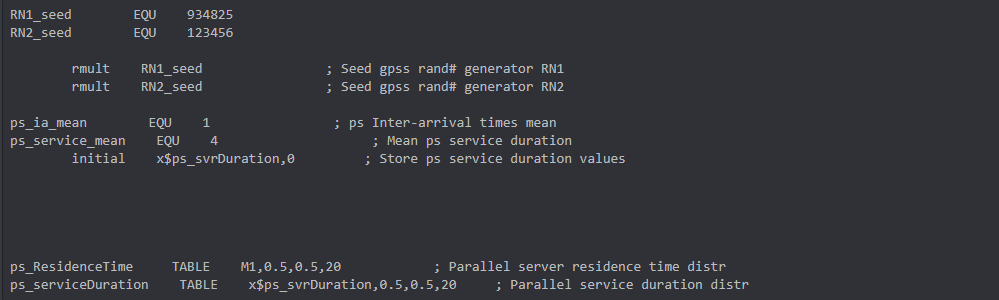
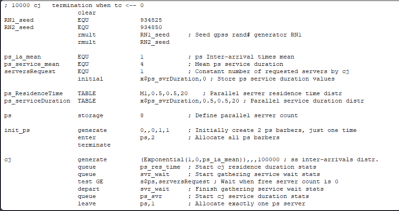
c) Be prepared to email to the grading account your source code, BUT ONLY IF REQUESTED TO DO SO DURING GRADING

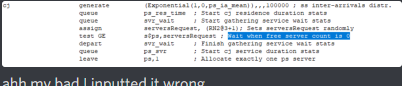
*====================================================================================*

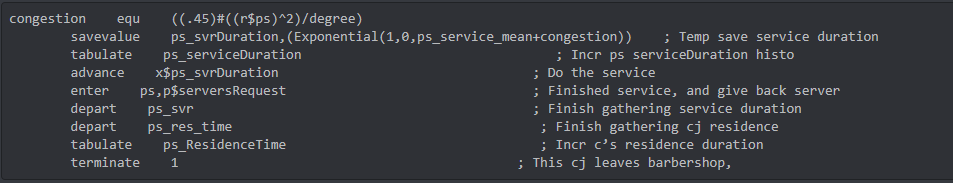
*Note - An overall hint/suggestion – you do not need to code 2 separate models for the 2 Configurations. You can comment out blocks & statements that are needed or not needed, per Configuration (See h1 solution as an example).*

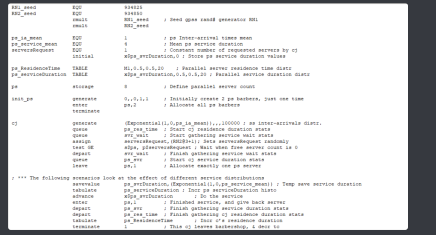
**

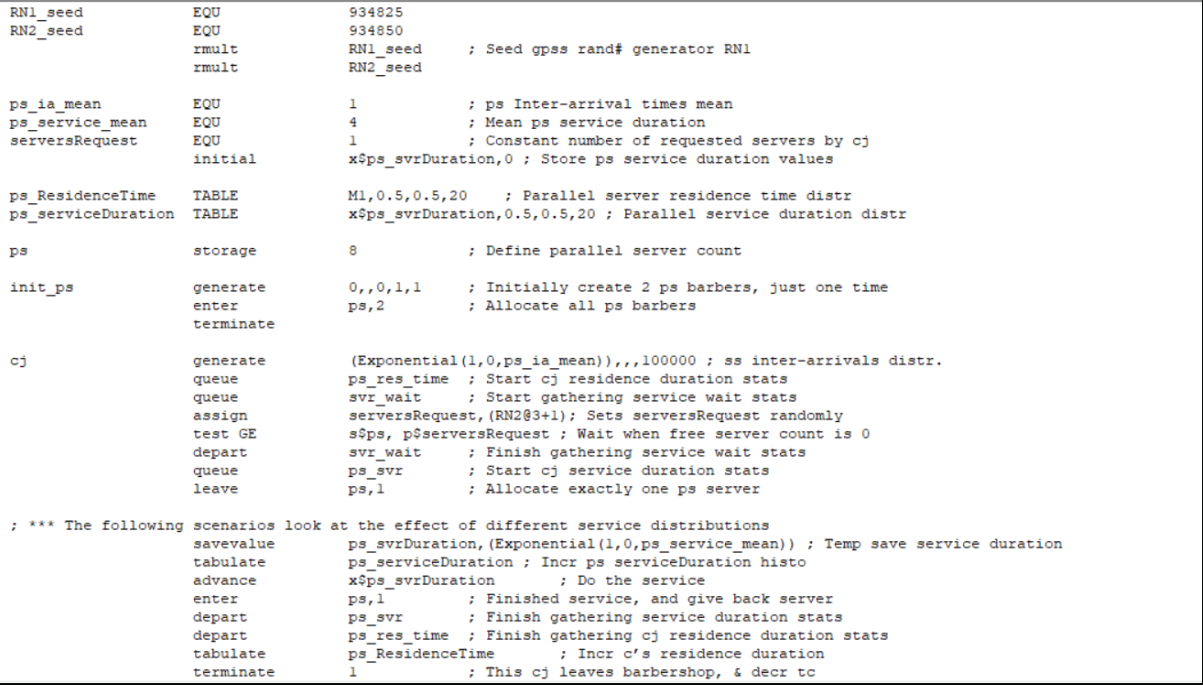
**

**

**

**

**

**