End Semester Examination

IIT Kharagpur, CSE Dept., Spring'16

(CS60058) FAULT TOLERANT SYSTEMS (Full marks = 100)

Answer all questions. In case of reasonable doubt, make practical assumptions

1. (a) Construct the decoding circuit for (15,11) cyclic code with generator polynomial X^4+X^3+1. Clearly explain how the circuit exactly performs the decoding. **[5]**

   (b) Describe the salient features of different RAID levels. **[10]**

   (c) Compute the MTTDL for RAID-1 starting from the Markov model and state equations. **[5]**

2. (a) Consider an NMR system that produces an eight-bit output where $N = 2m + 1$ for some m. Each processor fails at a constant rate λ and the failures are permanent. A failed processor produces any of the 2^8 possible outputs with equal probability. A majority voter is used to produce the overall output, and the voter is assumed never to fail. What is the probability that, at time t, a majority of the processors produce the same incorrect output after executing some program ? **[5]**

   (b) A processor suffers transient fault at a rate λ per unit time. The lifetime of a transient fault is exponentially distributed with parameter μ (which means the processor states changes from 'faulty' to 'working' at rate μ). For a software task, the processor implements fault-tolerance in the following manner. The processor executes the same task twice and if both executions are fault-free only then the output is really considered, otherwise it is discarded. The second execution instance starts τ time after the first execution instance starts. Each execution instance takes time s and $\tau > s$. Considering that the first execution instance started at some time T, compute the probability of the output being considered and not discarded. **[10]**

   (c) In a chordal (loop) network of n nodes, compute a value of skip distance which maximizes the network reliability. **[5]**

3. (a) Describe the version correlation model for software failure analysis and derive a suitable expression for computing the joint failure probability of two software versions. **[3]**

   (b) Consider the following programs as two different versions of the same software.

```
F1(int x, int y)
{
 int a=10, b=5, z;
 if(x>a && y<a)
 {
  if(y<b)
    z=<0.01> f1(x,y);
  else
    z=<0.02> f2(x,y);
 }
 else if(y>25)
    z=<0.001> f3(x,y);
 else
    x=<0.003> f4(x,y);
 x=<0.01> f5(z);
 return x;
}
```

```
F1'(int x, int y)
{
 int a=10, b=5, z;
 if(x>a && y<a)
  z=<0.005> f1'(x,y);
 else if(y>25)
 {
  if(x<=a)
    z=<0.001> f2'(x,y);
  else
    z=<0.003> f3'(x,y);
 }
 else
    x=<0.003> f4(x,y);
 x=<0.01> f5(z);
 return x;
}
```

Variable	Interval	Probability
x	[0,10]	0.7
	[10,40]	0.3
y	[0,5]	0.25
	[5,10]	0.6
	[10,40]	0.15

The quantities inside "<>" are the probabilities with which different function calls suffer failure in execution. The difference between the versions can be motivated as follows. Inside the region "$x > a \,\&\&\, y < a$", the first version uses two different functions (f1 and f2) inside two different subspaces. The functions perform the same job but it just happens that f1 has better numerical stability inside $y < b$ while f2 is better inside the other subspace. This motivates their choices. In the other version, we have f1' (an equivalent function) which is far more reliable (but slow due to

serial re-execution). Similarly, f3, f2' and f3' are functionally same but have different reliabilities. The probability distributions of inputs are given in the adjoining table. For example, the variable x has 0.7 of probability mass distributed uniformly in the range [0,10]. Compute failure probability of each individual version as well as the joint failure probability. **[2+2+2]**

(c) Consider the following program with the quantities inside "<>" being the probabilities with which different function calls suffer transient failure. The probability distribution of the input variables are stated similar to the previous problem.

```
system(int x,int y)
{
    int a=25, b=10, c=5, z;
    if(y > a)
        z=<0.10> f1 (x,y);
    else
        z=<0.09> f2 (x,y);
    if(x > b && y < c)
        z=<0.05> f3(x,y);
    else if (x<=b && y>=c)
        z=<0.07> f4(x,y);
}
```

Variable	Interval	Probability
x	[0,10]	0.65
	[10,40]	0.35
y	[0,5]	0.25
	[5,10]	0.35
	[10,25]	0.30
	[25,40]	0.10

F	$Ex_F(\mu S)$	C_F
f_1	2	0.85
f_2	3	0.80
f_3	10	0.90
f_4	8	0.87

i. Compute the overall reliability of the program. (Hint: there are four execution paths and overall reliability is the weighted sum of reliability of individual paths !!) **[7]**

ii. Let us now consider that for every function F, the execution time Ex_F is given in micro seconds and there exists a perfectly reliable fault detection hardware which detects with probability C_F (coverage factor) if function F has suffered a transient failure (rightmost table). Assume that apart from the functions, all other program statement executions take negligible time.
It is possible to re-execute a function F multiple times (say i) in order to increase system reliability. The idea is that in case of failure, the function can again be executed. Such repeated attempts can be made for a maximum of i number of times. In such a case, the failure of the $(i-1)$-th execution needs to correctly detected so that the i-th execution is attempted. If the failure is not detected, then no re-execution is performed leading to system failure. Consecutive i number of failures for F also lead to system failure. In a fault-tolerant version of the given program, every function can thus be labeled with some re-execute index i. However, in such a case the worst case end-to-end execution time of the system also increases. Considering a maximum end-to-end delay of 18 micro second in the worst case, provide re-execute index for each function so that the whole system reliability is at least 0.92. **[4]**

4. (a) Discuss the DFS based routing algorithm for injured Hypercubes. Elaborate on i) How does the algorithm select a detour when all optimal paths are blocked ? ii) What is the usefulness of backtracking in the algorithm? **[3+2+2]**

(b) In a SHIFT (Software Implemented Hardware Fault Tolerance) scheme, you are needed to choose a factor k for transforming (i.e. multiplying and finally dividing) the integer variables in a program to derive a second version of the program. Faults are of the type permanent stuck-at-0. What are the possible advantages and disadvantages of choosing $k = -1$? **[2]**

(c) In a Recovery Block structure comprising n software versions with a) failure probability of each software version = f, b) test sensitivity = s and c) test specificity = σ, derive an expression of the probability that the structure really provides an output thus making the scheme successful. **[3]**

(d) When running correctly, the output z of two different versions of the same software system have $f_1(z)$ and $f_2(z)$ as the probability density functions (assume L is some positive constant).

$$f_1(z) = \begin{cases} \dfrac{\mu_1 e^{-\mu_1 z}}{1 - e^{-\mu_1 L}} & \text{if } 0 \le z \le L \\ 0 & \text{otherwise} \end{cases} \qquad f_2(z) = \begin{cases} \dfrac{\mu_2 e^{-\mu_2 z}}{1 - e^{-\mu_2 L}} & \text{if } 0 \le z \le L \\ 0 & \text{otherwise} \end{cases}$$

The failure of the versions happen independently over the entire input space. For both the versions, in case of failure they output any value over the interval $[0, L]$ with equal probability. The failure probability for version 1 and 2 are q_1 and q_2 respectively. Consider that the two versions are connected using a Recovery Block scheme (version 1 is primary) where the acceptance test of version 1 is set for the interval $[0, \alpha_1]$ and the acceptance test of version 2 is set for the interval $[0, \alpha_2]$ such that $\alpha_1, \alpha_2 \leq L$. There are no further recovery stages in the system. The penalty for putting out an incorrect value is π_{bad} and the penalty for not producing any output at all is π_{stop}. Derive an expression for the expected total penalty incurred by the system. **[8]**

5. (a) Derive expressions for bandwidth, connectability and accessibility of a multi-stage Butterfly network with no extra (fault-tolerant) stage. State the advantages of a CCC network topology. **[3+3+3+2]**

   (b) In a time based software rejuvenation scheme, derive an expression for optimal rejuvenation period considering C_e as cost of failure, C_r as cost of each rejuvenation and $\lambda \times P^n$ as the expected number of failures in a time interval of size P. **[5]**

   (c) Explain the idea behind a Stack Smashing attack. What are the programming language features (or weakness if you call it so) exploited by attackers who perform such attacks ? How are such attacks mitigated in modern programming languages/compilers. **[2+1+1]**